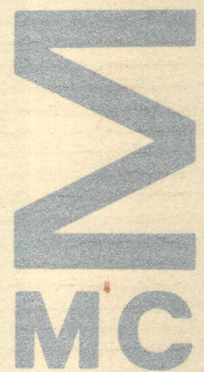
ma
the
ma
tisch

cen
trum



VAKANTIECURSUS 1973

VC 27/73 AUGUSTUS

ABSTRACTE INFORMATICA

amsterdam 1973

stichting
mathematisch
centrum

VAKANTIECURSUS 1973

VC 27/73 AUGUSTUS

ABSTRACTE INFORMATICA

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

VAKANTIECURSUS 1973

ABSTRACTE INFORMATICA

INHOUD

ABSTRACTE INFORMATICA

Algemene Inleiding

door

L.A.M. Verbeek

Het lijkt niet overbodig te vertellen wat abstracte informatica behelst. Nog afgezien van het abstracte hierbij is zowel de naam informatica als het erdoor aangeduide geheel van onderwerpen nogal nieuw. Bovendien komen in die onderwerpen grote veranderingen voor door de betrekkelijk snelle ontwikkeling van de inhoud en het belang van informatica.

Het woord informatica stamt van het Franse woord "informatique", dat elf jaar geleden werd bedacht als samentrekking van "information" en "automatique".

Als omschrijving van informatica gebruik ik graag de formulering, die o.a. staat in het memorandum [1], dat opgesteld is in verband met coördinatie van informatica in het wetenschappelijk onderwijs:

> "De informatica omvat de theoretische en praktische aspecten van de verwerking - in het bijzonder met behulp van automaten - van informatie, gezien als de formele neerslag van kennis en communicatie, op alle gebieden van wetenschap en samenleving".

Datzelfde memorandum geeft een meer concrete beschrijving van informatica door de volgende opsomming van onderwerpen, die samen het kerngebied van de informatica vormen:

> "informatiestructuren, algoritmen, programmeertalen
> organisatie en structuur van rekenautomaten
> systeemprogrammering
> informatiebeheer
> machinale intelligentie, simulatie".

Verder geeft het memorandum nog een lijst van vakgebieden, waarin de infor-- matica wordt toegepast en een lijst van wiskundige en technische vakgebieden, die de informatica ondersteunen.

Deze opsommingen geven een echte beperking aan van de geciteerde, meer algemene, omschrijving. Het gaat kennelijk om het werken met (digitale en niet analoge) computers en wat je daar allemaal bij nodig hebt of kunt gebruiken. Netjes gezegd: in het vakgebied informatica houdt men zich bezig met het bestuderen, ontwikkelen en toepassen van begrippen, verbanden tussen begrippen, methoden, werkwijzen en middelen, die van belang zijn bij het verwerken van informatie door computers of bij het ontwerpen van computers. Bij de theoretische aspecten gaat het om begripsvorming en inzicht in samenhang en methoden bij informatieverwerking door computers. Bij de practische aspecten van informatica gaat het om het maken en gebruiken van apparatuur en programmatuur, waarbij de nadruk, volgens de gegeven beschrijvingen, duidelijk op dit laatste ligt.

Voor we verder gaan moet eerst een andere kijk op informatica op zijn minst even aangestipt worden. Het overgrote deel van het gebruik en ook het directe maatschappelijk belang van computers ligt op het gebied van administratieve en bestuurlijke informatieverwerking. Dit verklaart waarom het vakgebied informatica ook wel wordt omschreven (zie bijvoorbeeld [2]) als:

> "de theoretische studie van informatiesystemen tezamen met hun praktische realisatie in menselijke taken en machines, speciaal in computers".

In de opvatting over informatica, die hierin wordt weergegeven, denkt men meer aan de toepassing van het gereedschap, in ruime zin, terwijl bij de eerder gegeven beschrijving het gereedschap zelf en het maken ervan juist op de voorgrond staat. We zullen ons hier verder niet meer bemoeien met toepassingen, maar ons concentreren op het gereedschap volgens de eerst gegeven beschrijving van informatica.

Om terug te komen op het abstracte van abstracte informatica kunnen we het volgende stellen. Abstracte informatica omvat die delen van informatica, waarbij wordt afgezien van concrete aspecten van apparatuur en programmatuur. Deze negatieve aanduiding is uiteraard niet erg verhelderend. Het helpt misschien iets, abstract te koppelen met theoretisch door

te stellen dat abstracte informatica de theoretische, dus niet praktische aspecten omvat van de informatica, gezien als wetenschapsgebied. Zinvoller is het enkele onderwerpen aan te duiden, die tot de abstracte of theoretische informatica gerekend worden.

Bij *informatiestructuren* denkt men aan de samenhang van de informatie zoals die in de computer, vooral in zijn geheugen, wordt gerepresenteerd. Uiteraard is die interne representatie erg belangrijk voor de verwerking van informatie door de computer. Dit geldt zowel voor de representatie van elementaire informatie, zoals een letter of een cijfer, als ook van meer gestructureerde informatie, zoals een getal, een paragraaf tekst, een lijst namen met adressen of een matrix met getallen als elementen.

Zoals voor ons rekenkundige bewerkingen gemakkelijker verlopen als de getallen in Arabische in plaats van Romeinse cijfers zijn gegeven, zo kan ook de ene representatie van informatiestructuren in de computer beter zijn dan de andere, afhankelijk van de soort bewerkingen, die de informatie moet ondergaan. Ook eventuele noodzaak om informatie over te brengen van intern- naar achtergrondgeheugen (magneetband of schijf) en het bewaren van informatie leidt tot zorgvuldig bekijken van de structurering van informatie en de representatie van informatiestructuren. Het bestuderen en onderzoeken van mogelijkheden van diverse wijzen van structurering en representatie van informatie in computers behoort tot de theoretische of abstracte informatica.

Over *algoritmen*, d.w.z. methoden die gevolgd worden om informatie te verwerken teneinde een of ander doel te bereiken, is veel te zeggen i.v.m. abstracte informatica. Zeer fundamentele en abstracte theorie over algoritmen is vanuit de mathematische logica naar de informatica gekomen met begrippen als berekenbaarheid en complexiteit. Ook in de informatica zelf is de studie van algoritmen, zoals deze in de dagelijkse praktijk gebruikt worden in allerlei programma's, tot bloei gekomen. Het bewijzen van uitspraken over correctheid, gelijkwaardigheid en complexiteit van algoritmen is, vooral de laatste jaren, onderwerp van veel abstract, theoretisch werk. Ook de structurering van programma's om dergelijke uitspraken over een algoritme, dat in een programma gestalte heeft gekregen, te kunnen doen of te vergemakkelijken, behoort tot de actuele en ook fundamentele onderwerpen van abstracte en theoretische informatica.

Programmeertalen dienen om gemakkelijker of beter gebruik te kunnen maken van de mogelijkheden tot informatieverwerking door computers. Behalve voor communicatie van mens naar machine worden elementen van programmeertalen en delen van programma's ook wel gebruikt voor communicatie tussen mensen, zoals in de vakliteratuur blijkt. Het op geschikte manier beschrijven van syntax en semantiek van een hele programmeertaal vergt veel inzicht in fundamentele begrippen van informatica. Dat geldt ook voor het passend beschrijven van begrippen, zoals recursie en formele of actuele parameters, die in programmeertalen voorkomen. Uiteraard horen studie en ontwikkeling van deze onderwerpen in de abstracte informatica thuis.

De hierboven genoemde onderwerpen uit de theoretische of abstracte informatica zijn sterk met elkaar verweven. De genoemde onderwerpen zijn, behalve in theoretisch en abstract opzicht, ook praktisch en concreet van belang in de informatica. Zo hoort het ook, de abstracte en theoretische informatica met zijn sterk mathematische en logische inslag vormt een integrerend deel van de informatica, die een technisch en toegepast wetenschappelijk karakter heeft.

Referenties

[1] Memorandum: *Coördinatie Informatica*,
uitgebracht 1971-07-05 door de Subcommissie Informatica van de Sectie Wiskunde van de Academische Raad.

[2] B.K.Brussaard, *Informatiesystemen in de practijk*,
inaugurale rede TH Delft, 1973-05-30, Uitgeverij Waltman, Delft.

TAAL EN AUTOMAAT

door

L.A.M. Verbeek

De theorie van formele talen en die van abstracte automaten vormen twee nauw verwante onderwerpen uit de theoretische informatica. Taal en automaat zijn hierin abstracte mathematische objecten, die model staan voor een deel van de essentie van programmeertalen en van computers of informatieverwerkingsprocessen. In het volgende zal eerst iets geschreven worden over formele talen, dan over abstracte automaten en tenslotte over het verband tussen die twee.

Wellicht ten overvloede moet hier nog met nadruk vermeld worden dat alleen een klein deel van de meer eenvoudige zaken uit de betreffende onderwerpen wordt geëtaleerd. Dit gebeurt in een vorm die een mengeling is van motivering voor en inhoud van de onderwerpen in de hoop dat zodoende de essentie duidelijk wordt.

1. Taal

1.1 Inleiding

Voor we op de theorie van formele talen ingaan zullen we eerst aangeven dat er vanuit drie vakgebieden, te weten algemene linguistiek, mathematische logica en informatica, een aanzet tot formele talen komt.

We beschouwen eerst gewone menselijke taal en beperken ons daarbij tot de geschreven vorm ervan, bijvoorbeeld geschreven Nederlands. Bovendien zien we af van geschreven versies van uitroepen en dergelijke en wel zó dat we alleen keurige volzinnen overhouden, die grammaticaal correct zijn opgebouwd.

Met dit deel van menselijke taal zijn de formele talen, die we straks zullen beschrijven, enigszins te vergelijken. Na deze inperkingen van de menselijke taal is het gemakkelijk de verschillende aspecten ervan aan te duiden, die voor de theorie van formele talen van belang zijn. We doelen op de samenstelling van geschreven taaluitingen, zoals zinnen, op de betekenis ervan en op het gebruik dat we ervan maken; d.w.z. op de syntactische, semantische en pragmatische aspecten.

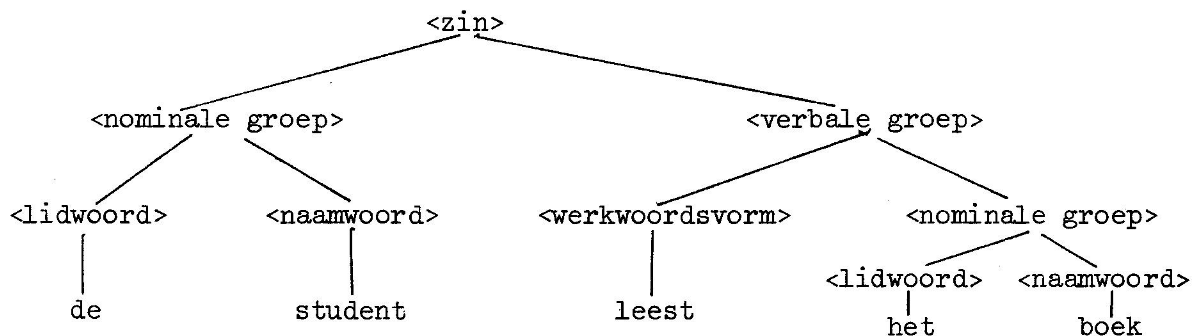
Formele talen komen ook voor in de formele systemen van de mathematische logica. Zo'n taal bestaat dan uit alle welgevormde formules. Dat zijn de eindige symboolrijen, die door het op bepaalde wijze toepassen van expliciet gegeven regels opgebouwd kunnen worden uit de elementaire tekens of symbolen van het systeem.
Als voorbeeld zouden we in de gewone rekenkunde de formule $(3+2)\times 5 = 18+4$ welgevormd (hoewel onjuist) noemen, terwijl de formule $3+\times 25 =) 1(8+5)$ daar niet welgevormd is.
Bij zinvolle interpretatie betekenen de welgevormde formules iets van belang en ze kunnen ook min of meer nuttig of doeltreffend zijn, afhankelijk van het beoogde doel. De syntactische, semantische en pragmatische kanten zijn er duidelijk in aanwezig.

De laatste twintig jaar zijn er computertalen, of liever, programmeertalen ontwikkeld om met digitale computers te werken. Daarbij kunnen we aan een programma in een programmeertaal denken als aan een zin in die taal. Zo'n programma moet correct zijn samengesteld om iets te betekenen voor de computer en verwerkt te kunnen worden. Het is een weergave van een algoritme, d.w.z. een verwerkingsproces, waarvan de te onderscheiden stappen en de manier van doorlopen van die stappen expliciet zijn aangegeven.
Een programma kan bij de gewenste bewerkingen op de bijpassende gegevens meer of minder doeltreffend of zuinig zijn. Ook bij programmeertalen komen dus weer syntactische, semantische en pragmatische kanten te voorschijn.

Het gemeenschappelijke van alle drie aangegeven soorten taal: geschreven menselijke taal, mathematisch-logische taal en programmeertaal zit in de syntactische, semantische en pragmatische aspecten, die tezamen de taal tot geschikt communicatiemiddel maken. Het zou erg fijn zijn als we zouden kunnen poneren dat de theorie van formele talen begrip en inzicht verschaft over deze drie aspecten van alle drie genoemde en eventueel ook

andere soorten taal. Maar dat is niet zo en zelfs helemaal niet zo. Wel geeft de theorie van formele talen inzicht in en begrip voor het beschrijven van o.a. programmeertalen en het automatisch verwerken van programma's. Bij formele talen staat vooral de syntactische kant op de voorgrond, maar ook komt iets van de semantiek en soms zelfs van de pragmatiek tot zijn recht.

Een formele taal zal worden gedefinieerd als een collectie zinnen, die opgebouwd zijn volgens bepaalde regels. Deze regels zijn gegeven in de grammatica van de betreffende taal. Om de gedachte te bepalen zullen we dit illustreren aan de Nederlandse zin "de student leest het boek". We kunnen deze zin ontleden of analyseren, bijvoorbeeld zoals in het volgende schema is aangegeven. De namen tussen < en > geven hierbij de soort aan van de betrokken taalcategorie.



Dit schema kan ook opgevat worden als een weergave van het ontstaan van de zin volgens onderstaande regels.

Begin met <zin> en schrijf daarvoor in de plaats <nominale groep> gevolgd door <verbale groep>. We duiden deze herschrijfregel aan door

<zin> → <nominale groep> <verbale groep>.

Vervolgens vervangen we <nominale groep> door <lidwoord> gevolgd door <naamwoord>, aangegeven door de herschrijfregel

<nominale groep> → <lidwoord> <naamwoord>.

Daarna passen we nog de volgende herschrijfregels toe

<verbale groep> → <werkwoordsvorm> <nominale groep>

<lidwoord> → de

<lidwoord> → het

<naamwoord> → student

<naamwoord> → boek

<werkwoordsvorm> → leest

De gegeven acht herschrijfregels vormen de grammatica, die geschikt is om de zin "de student leest het boek" te genereren. Met deze regels kunnen o.a. ook de on-zinnen "het student leest de boek" en "de boek leest de boek" worden gegenereerd.

Om de aangegeven gang van zaken bij het beschrijven van een taal met een grammatica te formaliseren hebben we enkele begrippen en enige afspraken over notatie nodig.

Een alfabet $V = \{v_1, v_2, \ldots, v_n\}$ is een eindige niet-lege verzameling van letters. We bekijken de verzameling V^+ van alle woorden, d.w.z. eindige rijen letters, samen met de binaire operatie van het aaneenschakelen van twee woorden. We noteren een woord door de samenstellende letters zonder scheidingstekens aaneen te schrijven. Als $w_1 = v_1v_2\ldots v_{n1}$ en $w_2 = v'_1v'_2\ldots v'_{n2}$ twee woorden zijn, dan is uiteraard ook de aaneenschakeling ervan, dus $w_1w_2 = v_1v_2\ldots v_{n1}v'_1v'_2\ldots v'_{n2}$ een woord.

(Verder geldt kennelijk voor elk drietal woorden w_1, w_2 en w_3 dat $(w_1w_2)w_3 = w_1(w_2w_3)$. Zodoende is V^+ gesloten onder het aaneenschakelen en die operatie is associatief, zodat V^+ een halfgroep, zelfs de vrije halfgroep over V, is.)

Elk woord w uit V^+ heeft een lengte $l(w)$. Het is evident dat voor alle w_1 en w_2 geldt $l(w_1w_2) = l(w_1) + l(w_2)$. Uit diverse overwegingen voegt men vaak aan V^+ nog het "lege" woord ε toe, met $l(\varepsilon) = 0$ en waarvoor geldt $\varepsilon w = w\varepsilon = w$ voor elke w uit $V^* = V^+ \cup \{\varepsilon\}$. ($V^*$ is de zgn. vrije monoïde over V.)

We zullen a^n schrijven voor de rij aa...a van n maal de letter a en we schrijven ook w^n voor de rij ww...w van n maal de rij w. Zodoende is bijvoorbeeld a^3b^3 = aaabbb en $(ab)^2$ = abab.

Met al deze afspraken gewapend kunnen we nu beginnen formele talen precies te beschrijven.

1.2 Generatieve grammatica en formele taal

Formele talen staan model voor o.a. programmeertalen. Ze worden gedefinieerd met behulp van generatieve grammatica's.

Definitie 1.1 Een generatieve grammatica G is een geordend viertal, $G = \langle V_h, V_e, P, Z\rangle$, waarin het hulpvocabulaire V_h en het eindvocabulaire V_e disjuncte alfabetten zijn, $Z \in V_h$ het zinsymbool is en P een eindige niet-lege verzameling productieregels is. Een productieregel is een geordend tweetal (ρ,σ) van woorden uit V^*, met $V = V_h \cup V_e$ en wordt geschreven als $\rho \to \sigma$.

Er wordt hier en ook verder nogal verwarrend met de paren namen: letter - woord, woord - zin en alfabet - vocabulaire omgesprongen. De oorzaak van die verschillende namen voor dezelfde dingen is de verschillende oorsprong in linguistiek en wiskunde van de betreffende begrippen.

Drie voorbeelden van generatieve grammatica's zijn als volgt.

Voorbeeld 1.1 $G_1 = \langle\{Z, B, C, D, E\},\{a, b, c\}, P_1, Z\rangle$ waarbij P_1 bestaat uit de volgende acht productieregels:

1. $Z \to aZBC$	3. $CB \to CD$	7. $B \to b$
2. $Z \to aBC$	4. $CD \to ED$	8. $C \to c$
	5. $ED \to BD$	
	6. $BD \to BC$	

Voorbeeld 1.2 $G_2 = \langle\{Z\}, \{a,b\}, P_2, Z\rangle$ waarbij $P_2 = \{Z \to aZb, Z \to ab\}$.

Voorbeeld 1.3 $G_3 = \langle\{Z, A, B\}, \{a,b\}, P_3, Z\rangle$ waarbij P_3 bestaat uit de vier productieregels: $Z \to aA$, $A \to a$, $A \to aB$ en $B \to bA$.

De manier waarop een generatieve grammatica een formele taal beschrijft is als volgt te definiëren.

Definitie 1.2 Gegeven is de generatieve grammatica $G = \langle V_h, V_e, P, Z\rangle$ en twee woorden ϕ en ψ uit V^*.

a) ϕ brengt ψ direct voort of ψ is direct af te leiden uit ϕ, geschreven als $\phi \Longrightarrow \psi$, als er woorden μ en ν in V^* zijn zó dat $\phi = \mu\rho\nu$ en $\psi = \mu\sigma\nu$ en $\rho \to \sigma$ is een productieregel van G.

b) ϕ brengt ψ voort of ψ is af te leiden uit ϕ, geschreven als $\phi \overset{*}{\Longrightarrow} \psi$, als er een rij $\zeta_0, \zeta_1, \ldots, \zeta_k$, met $k \geq 0$, van woorden uit V^* is zó dat $\phi = \zeta_0$ en $\zeta_i \Longrightarrow \zeta_{i+1}$, voor $0 \leq i \leq k-1$, en $\zeta_k = \psi$.

c) De taal L(G) gegenereerd door G is de verzameling van alle woorden uit V_e^* die afgeleid kunnen worden uit Z, dus $L(G) = \{w \mid w \in V_e^*, Z \overset{*}{\Longrightarrow} w\}$. De elementen van L(G) heten zinnen van de taal en de woorden uit V^* die afgeleid kunnen worden uit Z heten zinsvormen.

De gedefinieerde uitbreiding van $\rightarrow$ tot $\Longrightarrow$ en verder tot $\overset{*}{\Longrightarrow}$ is niets anders dan het gebruik van de productieregels in P om woorden uit elkaar af te leiden.
Volgens grammatica G_3 is bijvoorbeeld aabA direct af te leiden uit aaB, dus aaB $\Longrightarrow$ aabA (dit zijn beide zinsvormen), en ook bbAB $\Longrightarrow$ bbAbA (dit zijn geen zinsvormen). Verder is de zin aababa af te leiden uit aaB, dus aaB $\overset{*}{\Longrightarrow}$ aababa, immers aaB $\Longrightarrow$ aabA $\Longrightarrow$ aabaB $\Longrightarrow$ aababA $\Longrightarrow$ aababa. Zoals in het voorbeeld van "de student leest het boek" in paragraaf 1.1 al werd aangegeven, is dit idee om productieregels toe te passen om woorden af te leiden gebruikt om de taal die G genereert te definiëren. L(G) is immers het resultaat van alle mogelijke toepassingen van de productieregels tot alleen eindsymbolen overblijven en waarbij begonnen wordt met het zinsymbool Z.

Voorbeeld 1.4 Toepassen van definitie 1.2 op de voorbeelden 1.1, 1.2 en 1.3 levert, misschien met enige moeite

$L(G_1) = \{a^n b^n c^n \mid n \geq 1\}$

$L(G_2) = \{a^n b^n \mid n \geq 1\}$

$L(G_3) = \{a(ab)^n a \mid n \geq 0\}$

In plaats van de hierboven gegeven mechanistische interpretatie van het gebruik van de productieregels kunnen we dat ook in wiskundige terminologie gieten. De verzameling $P \subseteq V^* \times V^*$, of $\rightarrow$, is een binaire relatie op V^* en $\Longrightarrow$ is de links- en rechtsinvariante afsluiting van $\rightarrow$ met betrekking tot de operatie van de monoïde V^*. Verder is $\overset{*}{\Longrightarrow}$ de reflexieve en transitieve afsluiting van $\Longrightarrow$. De taal L(G) die G genereert is dan de deelverzameling

van V_e^* die met Z in de relatie $\overset{*}{\Longrightarrow}$ staat, dus $L(G) = \{w \mid Z \overset{*}{\Longrightarrow} w\} \cap V_e^*$.
Door de eerste, mechanistische, interpretatie van het genereren van L(G) uit G wordt de aandacht misschien duidelijker gevestigd op het proces dat nodig is om de productieregels van G toe te passen. We zullen dan ook steeds aannemen dat we over een denkbeeldig mechanisme beschikken dat, als we het een willekeurige grammatica G aanbieden, daaruit de bijbehorende taal L(G) genereert.
Met andere woorden, een denkbeeldig mechanisme waarin definitie 1.2 gestalte krijgt. Daardoor kunnen we generatieve grammatica's zien als een eindige manier om talen, d.w.z. deelverzamelingen van V_e^*, te beschrijven.
De beschrijving van talen met behulp van generatieve grammatica's is geschikt vanuit verschillende gezichtspunten, o.a. om een indeling te maken in soorten talen.

1.3 De Chomsky-hiërarchie

Eerst vermelden we dat de deelverzamelingen van V_e^*, die met behulp van generatieve grammatica's kunnen worden beschreven, precies alle recursief opsombare deelverzamelingen zijn. Dit betekent dat de beschrijvende kracht van generatieve grammatica's precies even groot is als die van partieel recursieve functies of van Turing-machines. Mede op grond van taalkundige overwegingen heeft Chomsky [1] drie soorten beperkingen op productieregels gegeven.
Een generatieve grammatica is van:
type 0 als er geen beperkingen zijn.
type 1 als elke productieregel de vorm $\eta_1 A \eta_2 \rightarrow \eta_1 \omega \eta_2$ heeft, met $A \in V_h$ en $\omega \in V^+$ en $\eta_1, \eta_2 \in V^*$. In dit geval wordt dus steeds één hulpsymbool herschreven tot een niet-leeg woord mits een bepaalde, eventueel lege, linker- en rechtercontext aanwezig is. Zie voorbeeld 1.1. Zo'n grammatica heet contextgevoelig of contextafhankelijk evenals de erdoor gegenereerde taal.
type 2 als elke productieregel de vorm $A \rightarrow \omega$ heeft, met $A \in V_h$ en $\omega \in V^+$. Zie voorbeeld 1.2. Zo'n grammatica heet contextvrij evenals de erdoor gegenereerde taal.
type 3 als elke productieregel de vorm $A \rightarrow a$ of $A \rightarrow aB$ heeft, met $A,B \in V_h$

en a $\in V_e$. Zie voorbeeld 1.3. Zo'n grammatica heet regulier (ook wel eindige-toestandsgrammatica) evenals de erdoor gegenereerde taal.

Bij gegeven V_e duiden we de klassen van generatieve grammatica's van type 0, 1, 2 en 3 met respectievelijk G_0, G_1, G_2 *en* G_3 en de klassen van erdoor gegenereerde talen met respectievelijk L_0, L_1, L_2 *en* L_3. Uit het bovenstaande en de voorbeelden 1.1, 1.2 en 1.3 volgt dat $G_0 \supset G_1 \supset G_2 \supset G_3$ en dat $L_0 \supseteq L_1 \supseteq L_2 \supseteq L_3$. Hierbij staat $\supset$ voor echte inclusie. Een stukje theorie van formele talen, dat we hier niet kunnen geven, laat zien dat ook geldt $L_0 \supset L_1 \supset L_2 \supset L_3$ en dat L_1 echt bevat is in de klasse van recursief beslisbare deelverzamelingen van V_e^*. Deze ordening van formele talen noemt men de Chomsky-hiërarchie.

Nu we zoveel aandacht besteed hebben aan de generatieve grammatica's, is het nodig iets te zeggen over varianten ervan om te vermijden dat het idee ontstaat dat we hier, in essentie, alles vertellen. Een van de mogelijke varianten is de geprogrammeerde grammatica [6]. We kunnen deze zien als een grammatica, waarbij elke productieregel bestaat uit een kern, die een gewone productieregel zoals van een generatieve grammatica is, voorzien van een eenduidig label L en van twee verzamelingen labels, een slaagverzameling S(L) en een faalverzameling F(L). Het genereren van L(G) uit de geprogrammeerde grammatica G gaat nu als volgt. Het afleidingsproces begint met het zinsymbool Z en met de eerste productieregel, dus die met label 1. Als bij een gegeven zinsvorm het toepassen van de productieregel met label L wordt geprobeerd en dit slaagt, respectievelijk faalt, dan wordt daarna geprobeerd een productieregel uit S(L),respectievelijk F(L), toe te passen. Een eenvoudig voorbeeld van een geprogrammeerde grammatica, waarin F(L) = Ø voor elke L, is als volgt:

Voorbeeld 1.5 Beschouw de geprogrammeerde grammatica G met $V_h = \{Z,A,B,C\}$ en $V_e = \{a,b,c\}$ en zinsymbool Z en de volgende productieregels:

L	kern	S(L)	F(L)
1.	Z → ABC	2,5	∅
2.	A → aA	3	∅
3.	B → bB	4	∅
4.	C → cC	2,5	∅
5.	A → a	6	∅
6.	B → b	7	∅
7.	C → c	∅	∅

Het afleiden van de zinnen abc en $a^2b^2c^2$ van de taal $L(G) = \{a^nb^nc^n \mid n \geq 1\}$ gaat als volgt: $Z \underset{1}{\Longrightarrow} ABC \underset{5}{\Longrightarrow} aBC \underset{6}{\Longrightarrow} abC \underset{7}{\Longrightarrow} abc$ en
$Z \underset{1}{\Longrightarrow} ABC \underset{2}{\Longrightarrow} aABC \underset{3}{\Longrightarrow} aAbBC \underset{4}{\Longrightarrow} aAbBcC \underset{5}{\Longrightarrow} aabBcC \underset{6}{\Longrightarrow} aabbcC \underset{7}{\Longrightarrow} aabbcc$.
Hierbij geven de getallen onder de pijlen van de directe afleidingsstappen de label aan van de gebruikte productieregel.

De taal L(G), die door G van voorbeeld 1.5 wordt gegenereerd, is dezelfde als die door de contextgevoelige grammatica G_1 van voorbeeld 1.1 wordt gegenereerd, hoewel de kern van elke productieregel van G van de contextvrije soort is.

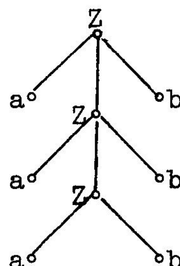
Bekijken we de generatieve grammatica $G' = \langle\{Z,A,B,C\},\{a,b,c\},P',Z\rangle$, waarin P' bestaat uit de zeven productieregels die we krijgen door de kern van die van G te nemen, dan blijkt $L(G') = \{a^kb^mc^n \mid k,m,n \geq 1\}$.
Het sturen van het afleidingsproces door de labels in geprogrammeerde grammatica's gebeurt niet bij generatieve grammatica's en daardoor worden andere soorten talen gegenereerd. De klasse L_{GC} van talen, gegenereerd door geprogrammeerde grammatica's met productieregels, waarvan de kern contextvrij is, bevat L_2 echt en is echt bevat in L_1, dus $L_1 \supset L_{GC} \supset L_2$.

Uit voorbeeld 1.5 blijkt dat het mechanisme, dat voor het toepassen van de productieregels zorgt, een essentieel deel is van de theorie over formele talen. Overigens zijn er ook beschrijvingswijzen van formele talen waarbij niet het generatieve idee wordt gebruikt. Voorbeelden hiervan zijn beschrijvingen via accepterende automaten; verder de algebraïsche beschrijvingen van contextvrije en van reguliere talen met behulp van stelsels vergelijkingen en ook de veel gebruikte beschrijvingen van reguliere talen met behulp van reguliere expressies. Dit zijn allemaal eindige formuleringen om eventueel oneindige talen te beschrijven.

1.4 Structuurbeschrijving

De beschrijving van een formele taal met een generatieve grammatica is bijzonder geschikt om aan elke zin van de taal ook een structurele beschrijving, de syntax, mee te geven in de vorm van de opbouw van die zin zoals die bevat is in de ontstaansgeschiedenis ervan.

We bekijken dit voor de contextvrije taal $L(G_2)$ van voorbeeld 1.2.
De zin a^3b^3 van die taal wordt als volgt uit Z afgeleid:
$Z \Longrightarrow aZb \Longrightarrow a^2Zb^2 \Longrightarrow a^3b^3$. De opbouw van deze zin kunnen we weergeven met de volgende zgn. afleidingsboom:

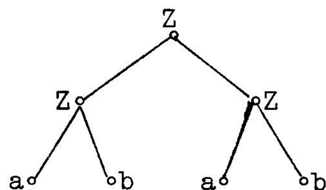


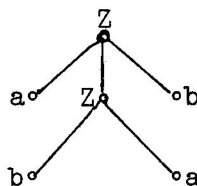
Zo'n afleidingsboom staat altijd op zijn kop met de wortel, gelabeld met het zinsymbool Z bovenaan. De zin wordt gevormd door de bladeren van de boom, die eindsymbolen als label hebben. De andere knooppunten van de boom hebben hulpsymbolen als label.
We kunnen meer zien van de betekenis van de structurele beschrijving van een zin aan het volgende voorbeeld.

Voorbeeld 1.6 $G = \langle\{Z\},\{a,b\},P,Z\rangle$ met als productieregels in P
$Z \rightarrow ab$, $Z \rightarrow ba$, $Z \rightarrow aZb$ en $Z \rightarrow ZZ$.
Drie verschillende afleidingen van de zin abab van L(G) zijn:

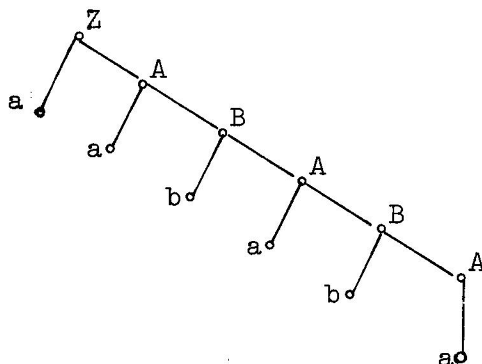
1. $Z \Longrightarrow ZZ \Longrightarrow abZ \Longrightarrow abab$
2. $Z \Longrightarrow ZZ \Longrightarrow Zab \Longrightarrow abab$
3. $Z \Longrightarrow aZb \Longrightarrow abab$

De bijbehorende afleidingsbomen voor 1. en 2., respectievelijk 3. zijn:

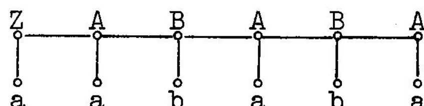




Het verschil tussen de afleidingen 1. en 2. is de volgorde van toepassing van de productieregels en dat is in de afleidingsbomen niet weer te geven. In afleiding 3. worden andere productieregels toegepast, zoals ook uit de afleidingsboom blijkt. De afleidingsboom van een zin geeft de opbouw van de afgeleide zin, zijn structuur, precies weer, terwijl triviale verschillen in het afleidingsproces, die bestaan uit verschillen in volgorde van toepassing van productieregels, worden genegeerd. Voor contextgevoelige grammatica's en nog meer voor type 0 grammatica's is de structuurbeschrijving niet zo eenvoudig als voor contextvrije grammatica's en we zullen er hier niets over zeggen. Bij reguliere grammatica's is de structuurbeschrijving van een zin altijd erg saai en de afleidingsboom kan in de vorm van een eenvoudige keten worden weergegeven. Bij de zin $a(ab)^2a$ van $L(G_3)$ van voorbeeld 1.3 is de afleidingsboom als volgt:



In de vorm van een keten weergegeven is dit:



We komen op de structurele aspecten, de syntaxis, van formele talen en het verband ervan met de semantiek terug, nadat iets over automaten is verteld.

Met de tot nu toe gegeven begrippen over formele talen is het mogelijk enkele toepassingen ervan globaal aan te geven.
De beschrijving van een taal door middel van een generatieve grammatica wordt steeds meer gebruikt om programmeertalen geheel of gedeeltelijk te bepalen. Een bekend voorbeeld daarvan is de programmeertaal ALGOL 60, die grotendeels is gedefinieerd door een contextvrije grammatica.
Een programma, geschreven in een programmeertaal, moet door een computer eerst worden geanalyseerd, d.w.z. ontleed in zijn samenstellende delen,

voordat de algoritme, die door het programma wordt gerepresenteerd, kan worden uitgevoerd. Veel praktisch gebruikte ontleders van programma's zijn gebaseerd op het inzicht en de methoden, die de formele taaltheorie voor ontleding geeft.
Ook buiten de informatica heeft de theorie van formele talen interessante toepassingen. Uiteraard is dat het geval in de linguistiek en met name in de transformationeel-generatieve taaltheorie.
Daarnaast worden varianten van generatieve grammatica's en de bijbehorende talen gehanteerd als modellen voor biologische groeiprocessen. Met name het ontstaan van biologische vormen, denk bijvoorbeeld aan bladeren van planten, kan op die manier worden gemodelleerd.

2. Automaat

2.1 Inleiding

Dertig jaar geleden publiceerden McCulloch en Pitts in een merkwaardig en inspirerend artikel [5] een wiskundig model voor de werking van het zenuwstelsel. Nader onderzoek van dit model leidde Kleene tot de formulering van een zgn. eindige automaat als wiskundig object [4].
Los hiervan, maar ongeveer gelijktijdig, werd de zgn. sequentiële machine geïntroduceerd, zie [3], als wiskundig model, dat diende als hulpmiddel bij het ontwerpen van sequentiële schakelingen in de electrotechniek.
Deze twee wiskundige modellen, eindige automaten en sequentiële machines, zijn in hun huidige formulering erg nauw verwant en het gemeenschappelijke ervan vormt een beschrijving van de werking van een mechanisme, dat een proces uitvoert waarvan het verloop mede gestuurd of bepaald wordt door een rij invoersymbolen. Welnu, een digitale computer kan, althans gedeeltelijk, ook gezien worden als een mechanisme, dat een proces uitvoert onder invloed van of gestuurd door een programma. Daarom zijn sommige delen van de werking van een computer te beschrijven en te begrijpen met behulp van automatentheorie. Het verband van automaten met formele talen komt ook nog aan de orde. We zullen hier alleen over de meest eenvoudige modellen, de sequentiële machines en eindige automaten, uitweiden. Ingewikkelder modellen, zoals stapelautomaten en Turing-machines, horen ook tot de theorie van abstracte automaten, maar daar is hier geen plaats voor.

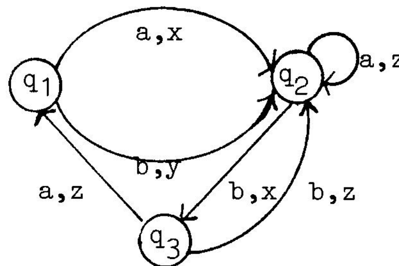
2.2 Sequentiële machines

Om het onderwerp precies aan te duiden geven we eerst

Definitie 2.1 Een sequentiële machine M is een geordend vijftal, $M = \langle I,U,Q,\delta,\lambda\rangle$, waarbij I, U en Q eindige niet-lege verzamelingen zijn van respectievelijk invoersymbolen, uitvoersymbolen en toestanden van M. Verder is $\delta: Q \times I \to Q$ de toestandsfunctie en $\lambda: Q \times I \to U$ de uitvoerfunctie van M.

Een sequentiële machine kan geheel gegeven worden door een transitiediagram. Zo'n diagram bestaat uit cirkeltjes verbonden door pijlen. De cirkels komen overeen met de toestanden, voor elke $q \in Q$ is er een cirkel gelabeld met q. Van cirkel q naar cirkel q' gaat een pijl gelabeld met i,u, waarbij $i \in I$ en $u \in U$, als $\delta(q,i) = q'$ en $\lambda(q,i) = u$. Een voorbeeld van zo'n sequentiële machine is als volgt.

Voorbeeld 2.1 $M_1 = \langle\{a,b\},\{x,y,z\},\{q_1,q_2,q_3\},\delta,\lambda\rangle$ waarbij δ en λ gegeven zijn in het volgende transitiediagram van M_1.



Wat er gebeurt, als aan een sequentiële machine niet een enkel invoersymbool maar een invoerwoord, d.w.z. een rij invoersymbolen, wordt toegevoerd, is eenvoudig uit het transitiediagram af te lezen. Bijvoorbeeld, als M_1 in begintoestand q_1 het invoerwoord aabbb krijgt, dan doorloopt M_1 de rij q_1,q_2,q_2,q_3,q_2,q_3 van zes toestanden (inclusief begin- en eindtoestand) terwijl het uitvoerwoord xzxzx is.
Evenzo levert het invoerwoord baba bij begintoestand q_2 het uitvoerwoord xzyz op. Wiskundig gezien wordt hierbij het domein van de functies δ en λ uitgebreid van $Q \times I$ tot $Q \times I^*$. We zullen ook het codomein van λ uitbreiden tot U^*.

Definitie 2.2 Gegeven is de sequentiële machine M = <I,U,Q,δ,λ>.
Voor elke $q \in Q$ en elke $i \in I$ en elke $w \in I^*$ definiëren we
$\delta(q,\varepsilon) = q$ en $\delta(q,iw) = \delta(\delta(q,i)w)$, zodat nu δ: $Q \times I^* \rightarrow Q$, en
$\lambda(q,\varepsilon) = \varepsilon$ en $\lambda(q,iw) = \lambda(q,i)\lambda(\delta(q,i),w)$, zodat nu λ: $Q \times I^* \rightarrow U^*$.
Verder definiëren we voor elke $q \in Q$ de werking M_q van M in toestand q als M_q: $I^* \rightarrow U^*$ met $M_q(w) = \lambda(q,w)$ voor elke $w \in I^*$.
Tenslotte definiëren we de werking M van M als $M = \{M_q \mid q \in Q\}$.

Bij voorbeeld 2.1 kunnen we een deel van M geven door de volgende tabel

w	$M_{q1}(w)$	$M_{q2}(w)$	$M_{q3}(w)$
ε	ε	ε	ε
a	x	z	z
b	y	x	z
aa	xz	zz	zx
ab	xx	zx	zy
ba	yz	xz	zz
bb	yx	xz	zx
aaa	xzz	zzz	zxz
¦	¦	¦	¦

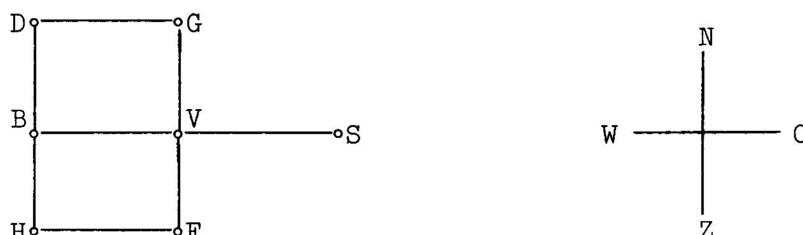
We zien dat een sequentiële machine beschouwd kan worden als een eindige manier om een eindige verzameling functies van I^* naar U^* te beschrijven. De werking M van een sequentiële machine M geeft precies het gedrag aan, d.w.z. alles wat uitwendig merkbaar is van de acties van M.
Het ontwerpen van een sequentiële schakeling bestaat in essentie uit het vinden van M, dus de interne constructie, als M is gegeven.
Bij gegeven I en U is niet elke eindige verzameling van afbeeldingen van I^* naar U^* te realiseren als werking van een sequentiële machine. Daartoe moet immers de lengte van het invoerwoord gelijk zijn aan die van het uitvoerwoord en bovendien de werking sequentieel zijn, d.w.z. de uitvoer symbool na symbool worden opgebouwd op basis van de invoer, die symbool na symbool wordt verwerkt.

Samenvattend kunnen we stellen dat een sequentiële machine een model is van mechanismen, die invoerwoorden omzetten in uitvoerwoorden, en dat de omzettingen, die door sequentiële machines worden beschreven, van beperkte soort zijn. We zullen hier niet verder ingaan op sequentiële machines, noch op varianten of uitbreidingen ervan.

2.3 Eindige automaten

We gaan ons nu bezighouden met eindige automaten, die o.a. gebruikt worden om de gang van zaken aan te geven bij het volgen van eenvoudige procedures en om te controleren of zo'n procedure wel juist is, d.w.z. van de gegeven beginsituatie leidt tot de gewenste eindsituatie.

Stel dat een kind met een gulden van huis H naar de snoepwinkel S gaat om twee pakjes suikervrije kauwgom te kopen. Het stratenplan is in onderstaande figuur aangegeven:



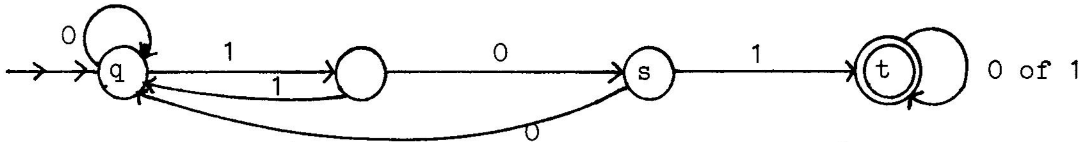
Het kind kan langs de bakkerswinkel B en de verfwinkel V, maar ook via de friteskraam F en V, naar S gaan. Een omweg langs de drogisterij D en de groentewinkel G is ook mogelijk. Het is duidelijk dat het kind van H naar S komt door eerst een straat noordwaarts en dan twee straten oostwaarts te gaan, aangeduid met noo. Ook ono, nnozo, onwnozo etcetera geven wegen van H naar S aan. Zonder enig begrip van de plattegrond als geheel kan het volgen van een rij instructies van dit type iemand van elk beginpunt naar elk gewenst eindpunt brengen. Maar dit is net wat in een sequentiële machine ook gebeurt, als we van de uitvoer afzien en een vaste begintoestand en gewenste eindtoestand(en) aangeven. Een wiskundig model voor het na elkaar ondernemen van een rij acties of voor het toetsen of een rij acties tot een gewenst doel leidt is als volgt.

Definitie 2.3 a) Een eindige automaat A is een geordend vijftal, $A = \langle I,Q,\delta,q_b,F\rangle$, waarbij I en Q eindige niet-lege verzamelingen zijn van respectievelijk invoersymbolen en toestanden. Verder is $\delta: Q \times I \to Q$ de toestandsfunctie, terwijl $q_b \in Q$ de begintoestand is en $F \subseteq Q$ de verzameling van accepterende toestanden is. Evenals in definitie 2.2 wordt δ uitgebreid tot $\delta: Q \times I^* \to Q$.

b) De automaat A accepteert een invoerwoord $w \in I^*$ als en alleen als $\delta(q_b,w) \in F$. De verzameling van door A geaccepteerde invoerwoorden is $T(A) = \{w \mid w \in I^*, \delta(q_b,w) \in F\}$.

Een eindige automaat is volledig te geven door een transitiediagram als dat van een sequentiële machine, waarin alles wordt weggelaten wat betrekking heeft op uitvoer en uitvoerfunctie. Wel moeten de begintoestand en de accepterende toestanden worden aangegeven, bijvoorbeeld door een dubbele pijl en dubbele cirkel, zoals in het volgende voorbeeld.

Voorbeeld 2.2 $A_1 = \langle\{0,1\},\{q,r,s,t\},\delta,q,\{t\}\rangle$ waarbij δ is gegeven in het volgende diagram



Na enig proberen zal blijken dat $T(A_1) = \{w101w' \mid w,w' \in I^*\}$ of, in woorden, A_1 accepteert alle invoerwoorden waarin de rij 101 voorkomt.

Hierboven gaven we als motivatie voor het bekijken van een eindige automaat het volgen van een procedure en het daardoor verifiëren van het totale effect van de procedure. We kunnen dit ook als volgt formuleren.
De betekenis van een eindige automaat A met invoerverzameling I is het beschrijven van $T(A) \subseteq I^*$ en wel op mechanistische, constructieve manier. Immers A is een model van een mechanisme, dat een invoerwoord $w \in I^*$ al of niet accepteert en zo de karakteristieke functie van T(A) realiseert.
Men kan zich afvragen welke deelverzamelingen van I^* door eindige automaten kunnen worden gekarakteriseerd, dus eindig-acceptabel zijn en welke niet. Het antwoord op deze vraag is op verschillende manieren te geven. Eén daarvan geeft direct het verband tussen eindige automaten en reguliere grammatica's.

3. Taal en automaat

We vallen met de deur in huis door direct het verband tussen eindige automaten en reguliere grammatica's te geven. Elke eindig-acceptabele deelverzameling van I^* is een reguliere taal over I. Omgekeerd is ook elke reguliere taal over een alfabet I een eindige-acceptabele deelverzameling van I^*. Anders gezegd: het karakteriserend vermogen van eindige automaten is gelijk aan dat van reguliere grammatica's. We moeten hier nog aan toevoegen dat deze gelijkwaardigheid ook constructief is aan te tonen. Bij een gegeven eindige automaat A is uit A een reguliere grammatica G te construeren zó dat L(G) = T(A). Ook is bij gegeven reguliere grammatica G een eindige automaat A te construeren uit G zó dat T(A) = L(G). Bij die constructie is het eindvocabulaire van G gelijk aan de invoerverzameling van A en het zinsymbool Z van G juist de begintoestand van A. Verder komen de hulpsymbolen van G globaal overeen met de toestanden van A, terwijl de productieregels de toestandsfunctie en de accepterende toestanden bepalen. Dit is in het volgende voorbeeld aangegeven.

Voorbeeld 3.1 Gegeven zijn de reguliere grammatica G en de eindige automaat A'. G is de grammatica van voorbeeld 1.3.

G = <{Z,A,B},{a,b},P,Z>

met P bestaande uit

Z → aA

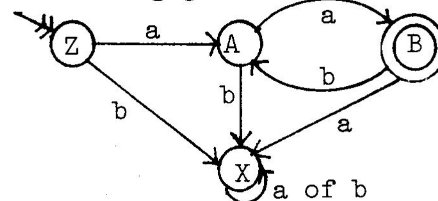
A → a

A → aB

B → bA

A' = <{a,b},{Z,A,B,X},δ,Z,{B}>

met δ als gegeven in het diagram

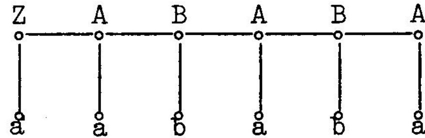


Het is instructief om te verifiëren dat $L(G) = T(A) = \{a(ab)^n a \mid n \geq 0\}$.

Volledige algoritmen voor de overgang van reguliere grammatica naar eindige automaat en omgekeerd zijn iets gecompliceerder, maar niet essentieel anders, dan het simpele voorbeeld 3.1 wellicht suggereert. Het is niet zinvol hier nader op details in te gaan.

Met de reguliere grammatica en de eindige automaat als gereedschap om reguliere talen te bepalen zijn velerlei stellingen te bewijzen over de klasse R van reguliere talen. Bijvoorbeeld dat R een Boole-algebra van deelverzamelingen van I^* vormt.

De karakterisering van reguliere talen met reguliere grammatica's en met eindige automaten zijn beide indirect, omdat ze als genererend en accepterend mechanisme de taal zin voor zin en niet als geheel bepalen. Zoals we in paragraaf 1.4 al zagen geeft een grammatica juist hierdoor bij elke zin, die wordt gegenereerd, een structurele beschrijving. Voor de zin aababa in voorbeeld 3.1 is die structuur gegeven door de volgende afleidingsboom in de vorm van een keten.



Ook elke eindige automaat geeft bij het accepteren van een zin de structuur ervan aan in de vorm van het accepterend proces, dat de automaat doorloopt. Wanneer we bijvoorbeeld de eindige automaat A' van voorbeeld 3.1 met begintoestand Z het invoerwoord $a(ab)^2a$ geven, zullen de toestanden Z, A, B, A, B, A in deze volgorde worden doorlopen, waarna de accepterende toestand B wordt bereikt. We zien dat de automaat A', die bij de grammatica G past, de zin accepteert en door het proces van het accepteren de zin precies analyseert wat betreft zijn structuur. De syntax van de taal wordt zowel door de grammatica als door de automaat weergegeven.

We kunnen hier nog iets meer aan vastknopen. Het accepteren van een zin door de automaat is een proces, dat precies verloopt zoals het proces van het genereren van die zin volgens de bij de automaat passende grammatica. Bovendien is de zin de drager van het betreffende proces. Hier zien we een demonstratie van het basisidee van de taal als communicatiemiddel. De productieregels, waarmee de zin is opgebouwd, worden stap voor stap door de automaat geanalyseerd. Als nu bij elke productieregel een bepaalde gewenste en bedoelde actie hoort, kan deze actie door de automaat worden gestart als hij in de betreffende toestand komt.

Het geheel van acties, uitgedrukt in de zin van de taal, wordt dan door de automaat in beweging gezet in de gewenste en bedoelde volgorde. We kunnen hiervoor ook lezen: het geheel van acties, uitgedrukt in het programma van de programmeertaal, wordt door de computer uitgevoerd. Hiermee is het verband tussen taal, of liever grammatica, en automaat wat betreft syntax en semantiek in essentie weergegeven.

We hebben bij automaten en het verband tussen taal en automaat alleen over eindige automaten en reguliere grammatica's geschreven. Voor contextvrije grammatica's speelt de zgn. stapelautomaat dezelfde rol als de eindige automaat doet voor reguliere grammatica's. Bij veel algoritmen voor informatieverwerking speelt een stapelmechanisme een essentiële rol. Toch gaan we hier niet in op stapelautomaten, maar verwijzen naar bijvoorbeeld [2], waarin ook veel andere onderwerpen uit de theorie van talen en automaten worden behandeld.

Referenties

[1] Chomsky, N., *On certain formal properties of grammars,* Information and Control, 2 (1959), 137-167.

[2] Hopcroft, J.E. and Ullman, J.D., *Formal languages and their relation to automata,* Addison-Wesley Series in Computer Science/I.P., Addison-Wesley Publ.Co., Reading etc., (1969).

[3] Huffman, D., *The synthesis of sequential switching circuits. Part I,* J. Franklin Inst., 257 (1954), 161-190.

[4] Kleene, S., *Representation of events in nerve nets and finite automata,* Research Memorandum RM-704, The Rand Corp., Santa Monica, Cal., (1951). Opgenomen in: Automata Studies, Ed.Shannon, McCarthy, Annals of Mathematical Studies no.34, Princeton Univ.Press, (1956).

[5] McCulloch, W. and Pitts, W., *A logical calculus of the ideas immanent in nervous activity,* Bull. Math. Biophys., 5 (1943), 115-133.

[6] Rosenkrantz, D., *Programmed grammars and classes of formal languages,* J. Assoc. Comput. Mach., 6 (1969), 107-131.

COMBINATOREN EN LAMBDAVORMEN

door

W.L. van der Poel

Samenvatting

In deze voordrachten zal een overzicht worden gegeven van de theorie van de combinatorische logica als fundering voor de lambdacalculus. In het bijzonder kunnen hiermee het getalbegrip, parametermechanismen, conditionele vormen en recursieve vormen begrepen worden vanuit een bijzonder eenvoudig gezichtspunt. Met slechts twee basisoperatoren wordt de gehele theorie opgebouwd.

1. Combinatoren

De combinatorische logica is een tak van de mathematische logica die oorspronkelijk is opgezet door Schönfinkel en Curry omstreeks 1926 om beter vat te krijgen op het begrip "variabele", i.h.b. "gebonden variabele". Om bijvoorbeeld aan te geven dat we eerst een functie g en daarna een functie f moeten toepassen kunnen we schrijven: f(g(x)). Alleen om de groepering met haakjes goed te kunnen aangeven hebben we de haakjes nodig. We kunnen ons behelpen door het "dot product" en schrijven f o g. In de combinatorische logica schrijven we Bfg. De combinator B maakt het functioneel product. Moeilijker wordt het als we willen aangeven dat we het resultaat van f(x) weer willen toepassen op g(x). We kunnen schrijven f(x) (g(x)), maar hierin speelt x eigenlijk geen rol. Met combinatoren kunnen we schrijven: Afg.

De combinatorische logica gaat er van uit dat er een verzameling van objecten bestaat, waarvan de aard niet nader zal worden aangeduid. Men zou

deze objecten kunnen zien als argumenten, maar ook als functies of operatoren. Tussen deze objecten zal gelden dat elk object toegepast kan worden op een ander object. Het resultaat zal weer een object uit de verzameling zijn. We zullen objecten aanduiden met identifiers (meestal enkele letters). Omdat het toepassen van een object op een ander object de enige operatie is, hebben we hiervoor geen apart teken nodig. We zullen schrijven F X of (F X) voor de toepassing van object F op object X. De haakjes dienen alleen om op de gebruikelijke wijze groepering aan te geven. De toepassing zal steeds betrekking hebben op één object. Als geschreven wordt (F X Y) dan bedoelen we: pas eerst F op X toe en het resultaat daarvan op Y. Deze uitdrukking zal dus links-associatief zijn. We hadden ook kunnen schrijven ((F X)Y). Een rechts-associatieve uitdrukking moet weergegeven worden met haken bijv.: (F(G X)) betekent: pas eerst G op X toe en pas daarna F op dat resultaat toe. Dit suggereert sterk de overeenkomst met f(g(x)) zoals in de normale wiskunde wordt geschreven.

Er zullen in de verzameling twee objecten zijn met bijzondere eigenschappen, genaamd A en K. Het zal blijken dat met behulp hiervan een groot aantal fundamentele begrippen uit de informatica verklaard kan worden. Deze operatoren of combinatoren verrichten slechts wat eenvoudige omzettingen in rijen van tekens en toch zijn zij voldoende om de gehele theorie van de recursieve functies te verklaren. We zullen deze zaken hier niet benaderen van een existentieel of axiomatisch standpunt, maar meer van een constructivistisch standpunt. We zullen eisen dat alle formules ook met zuiver mechanische middelen door een computer afgeleid of geëvalueerd kunnen worden. In feite zijn alle formules door een computer geproduceerd. Het is zelfs zo, dat hoewel de presentatie ook geheel met de hand gedaan kan worden, vele van de hier gebrachte resultaten alleen gevonden konden worden met behulp van de computer. Uit deze mechanisch verkregen resultaten kon dan achteraf inzicht worden verkregen. Bijvoorbeeld zal een geheel nieuwe wijze van getalrepresentatie worden gegeven met verrassend eenvoudige operatoren. De weg waarlangs deze operatoren gevonden zijn is echter zeer ingewikkeld geweest en had zonder computer niet kunnen gebeuren.

Wij komen nu tot de definities van de grondoperatoren A en K.

(A X Y Z) = (X Z(Y Z)) [1]

(K X Y) = X [2]

Daarin zijn X, Y en Z willekeurige objecten (dus bijv. ook A of K). De regels kunnen strikt mechanisch geïnterpreteerd worden. Een vorm, die begint met A en gevolgd wordt door drie argumenten, zal de in formule [1] gegeven omzettingen doen. Het is hier dienstig om er op te wijzen dat er nog meer argumenten kunnen volgen, die echter in eerste instantie niet meedoen. Bijv.

(A A X Y Z) = (A Y(X Y)Z) = (Y Z(X Y Z))

De operator K is de constantheidsoperator, de "wegmaker". Wat ook het tweede argument is, het komt in het resultaat niet meer voor. Dit wegmaken van informatie is een van de meest fundamentele handelingen, die er zijn. Bij alle logica is sprake van oorzaak en gevolg. Een omkeerbare operatie heeft eigenlijk niets gedaan. Als men c = a + b heeft, dan kan men uit het resultaat en een van de termen de andere weer terugvinden. Er is dus geen informatie verloren gegaan en er is ook niets bijgekomen. Als men evenwel een nieuw woord in een geheugen schrijft, gaat het oude woord verloren. Dit proces is onomkeerbaar evenals een redenering van oorzaak naar gevolg.

De operator A zou men de associator kunnen noemen. Hij doet drie dingen:

a) Hergroeperen. Y en Z worden tussen haakjes geplaatst.

b) Dupliceren. Er komen in het resultaat twee Z's voor.

c) Verwisselen. In het resultaat staat er een Z voor de Y.

In de literatuur komt men wel S tegen voor deze operator. Wij zullen Rosenbloom volgen met A, zodat we S later vrij hebben voor de successorfunctie.

We zullen ons op het puur formalistische standpunt stellen dat aan deze operatoren A en K geen nadere betekenis behoeft te worden toegekend dan als combinatoren in een rij van objecten, waarmee uitsluitend hergroepering, duplicering, verwisseling en annihilatie kan worden uitgevoerd. Dat deze fundamentele operaties afgebeeld kunnen worden op bepaalde zaken uit de menselijke begripssfeer is een prettige bijkomstigheid.

Hoewel alle verdere operatoren kunnen worden uitgedrukt in A en K alleen is het toch voordelig om een aantal andere operatoren in te voeren. We beginnen met de definitie van I:

I = (A K K) [3]

We kunnen nu aantonen, dat I de eigenschappen van de identiteitsoperator heeft.

(I X) $\underset{3}{=}$ (A K K X) $\underset{1}{=}$ (K X(K X)) $\underset{2}{=}$ X [4]

I toegepast op een willekeurig object X levert dat object X zelf op. We hadden als definitie net zo goed kunnen nemen: I = (A K Z) met [4a] willekeurige Z. Immers (I X) = (A K Z X) = (K X(Z X)) = X. De Z valt er toch uit.

De volgende operator zal gedefinieerd worden als:

B = (A(K A)K) [5]

Hiermee kunnen haakjes gezet worden (of omgekeerd ook weggewerkt worden).

(B X Y Z) $\underset{5}{=}$ (A(K A)K X Y Z) $\underset{1}{=}$ (K A X(K X)Y Z) $\underset{2}{=}$ (A(K X)Y Z) $\underset{1}{=}$
= (K X Z(Y Z)) $\underset{2}{=}$ (X(Y Z)) [6]

Het dupliceringselement en het verwisselen is hier uitgeschakeld en alleen het hergroeperingselement is behouden.

Toegepast op één object geeft B ook een zeer vaak bruikbare formule

(B X) = (A(K A)K X) = (A(K X)) [7]

Het zuivere dupliceringskarakter kan bereikt worden met:

W = (A A(A K)) [8]

Passen we dit weer toe op twee argumenten dan krijgen we:

(W X Y) = (A A(A K)X Y) = (A X(A K X)Y) = (X Y(A K X Y)) =
= (X Y(K Y(X Y))) = (X Y Y) [9]

Ook W toegepast op één argument levert een nuttige formule

(W X) = (A A(A K)X) = (A X(A K X)) = (A X I) [10]

het laatste ingevolge formule [4a].

Ter verificatie passen we het resultaat van deze laatste formule toe op Y en krijgen:

(A X I Y) = (X Y(I Y)) = (X Y Y)

Men kan aan de W combinator de volgende aanschouwelijke betekenis verbinden. Als * de vermenigvuldigingsoperator zou zijn, dan is (* X Y) het

product van X en Y. Alle functies zijn hier natuurlijk prefixoperatoren. Dan is (W *) de kwadrateringoperator, want (W * X) = (* X X).

W verdubbelt het tweede argument. Een functie om het eerste argument zelf te verdubbelen krijgen we door (W I). Immers
(W I X) = (I X X) = (X X) [11]

Een verwisselingsoperator krijgen we door te nemen:
C = (A(B B A)(K K)) = (A(A(K(A(K A)K))A)(K K)) [12]
Passen we dit toe op drie argumenten, die weer willekeurig zijn dan krijgen we:
(C X Y Z) = (A(B B A)(K K)X Y Z) =
(B B A X(K K X)Y Z) =
(B(A X)K Y Z) =
(A X(K Y)Z) =
(X Z(K Y Z)) =
(X Z Y) [13]
Tweede en derde argument zijn dus verwisseld. Ook hier kunnen de formules voor C, toegepast op één of twee argumenten nuttig zijn.
(C X) = (B(A X)K) [14]
(C X Y) = (A X(K Y)) [15]
De afleiding wordt aan de lezer overgelaten.

Een operator om twee argumenten te verwisselen krijgen we door (C I)
(C I X Y) = (I Y X) = (Y X) [16]

2. Extensionaliteit

Om een aantal stellingen af te kunnen leiden voeren we het beginsel van de extensionaliteit in:

Als voor alle X geldt, dat (F X) = (G X), dan is F = G [17]

Voor een constructivist is dit een gevaarlijk axioma. Het is immers niet mogelijk voor "alle" X te verifiëren of FX = GX. Omdat we geen bijzondere veronderstellingen hebben gemaakt omtrent X nemen we intuïtief aan dat het dan zal gelden voor alle X. Het is mogelijk de gehele theorie op te bouwen zonder gebruik te maken van dit axioma maar daarvoor in de plaats enkele hierna af te leiden stellingen als axioma te nemen.

De bewijzen worden dan wel veel bewerkelijker en leiden tot hetzelfde resultaat, zodat we achteraf toch kunnen rechtvaardigen dit eenvoudiger extensionaliteitsaxioma te hanteren.

Zo kunnen we nu bewijzen dat

(A K) = (K I) [17a]

Bewijs: (K I X Y) = (I Y) = Y

(A K X Y) = (K Y(X Y)) = Y

Beide combinatoren toegepast op willekeurige X en Y geven hetzelfde resultaat, dus zijn ze volgens [17] gelijk. Blijft nog de vraag aan welke vorm wij als meest primitieve de voorkeur moeten geven. Als leidend beginsel zouden wij kunnen hanteren, dat het aantal objecten in de vorm zo klein mogelijk is. Maar voor grote vormen zijn er vaak een aantal equivalente en even lange formuleringen. Het blijkt, dat hier een simplificatiebeginsel (net als in de theorie van het vereenvoudigen van algebraïsche vormen) bijzonder moeilijk is te formuleren op een zodanige wijze dat het ook voor een machine is te hanteren.

We zullen in deze paragraaf een aantal stellingen met en zonder bewijs afleiden, die we later nodig kunnen hebben.

(A(K X)I) = X [18]

Bewijs: (A(K X)I Y) = (K X Y(I Y)) = (X Y)

(A(A(K K)X)Y) = X [19]

Bewijs: (A(A(K K)X)Y Z) = (A(K K)X Z(Y Z)) = (K K Z(X Z)(Y Z)) =

= (K(X Z)(Y Z)) = (X Z)

Deze stellingen kunnen goede diensten bewijzen bij het vereenvoudigen van uitdrukkingen.

(A(K X)(K Y)) = (K(X Y)) [20]

Bewijs: (A(K X)(K Y)Z) = (K X Z(K Y Z)) = (X Y) = (K(X Y)Z)

Bij de volgende stellingen wordt het bewijs aan de lezer overgelaten. Het principe is steeds om zoveel objecten toe te voegen tot de definitie van de operatoren gebruikt kan worden. Aan het eind van de herleiding moet men dan dezelfde toegevoegde variabelen in dezelfde volgorde rechts in het resultaat overhouden.

(A(K I)) = I [21]

(A B(K I)) = I [22]

(B A(B K)) = K [23]

(B I) = I (direct gevolg van [21]) [24]

(B C K) = (B K) [25]

(B A(A(K K)) = K [26]

(A(B B B)(K K)) = (B(B K)I) [27]

(A(B(B B(B A(B A))))A) = (B(B(B A)B)(B A)) [28]

Zonder bewijs zij hier vermeld dat ook de formules [21], [22], [25], [26] en [28] als axioma genomen kunnen worden. Ook andere keuzen zijn nog mogelijk.

3. Het getalbegrip

De tot zover geleerde combinatoren zullen we nu gaan gebruiken om het getalbegrip te ontwikkelen. Als we de natuurlijke getallen beschouwen als operatoren, dan kunnen we opmerken, dat deze operatoren altijd werken op twee objecten, nl. de aard van het getalde object en de handeling, die er op wordt verricht. We kunnen spreken over twee schapen, twee tafels, twee stoelen, maar in het begrip twee komen deze tafels en stoelen niet meer voor. Ook de handeling om de te tellen objecten af te beelden op de getallen, de telhandeling, komt in het getalbegrip zelf niet meer voor. Of we nu twee schapen tellen of twee strepen turven, het begrip twee is invariant ten opzichte van het tellen of het turven. Zo kunnen we dus stellen dat de operator 2 toegepast op het tellen van schapen zal geven:

(2 TEL SCHAAP) = (TEL(TEL SCHAAP))

Er wordt een schaap geteld en er wordt nogmaals geteld. De operator 1 zal dus geven (1 TEL SCHAAP) = (TEL SCHAAP). Dit geeft een hint voor wat we kunnen kiezen als de operator 1, nl. I. Immers (I X Y) = (X Y).

We zullen daarom stellen 1 = I [29]

De operator 0 zal moeten leveren: (0 TEL SCHAAP) = (SCHAAP).
Er wordt immers niet geteld. De operator die dit doet is (K I). We zullen straks een systematische methode vinden om met behulp van lambdavormen zo'n operator op te sporen. Hier volstaan we met het bewijs:

(K I X Y) = (I Y) = Y [30]

De operator 2 zal moeten leveren

(2 X Y) = (X(X Y))

We zullen dit op de volgende manier herleiden:

(X(X Y)) = (B X X Y) = (W B X Y)

De haakjes zijn weggewerkt met B, de dubbele X met W. Daar X en Y weer willekeurig waren kunnen we concluderen

2 = (W B) [31]

We zullen dit nog iets verder herleiden om te zien of we het verband met 1 kunnen vinden om zodoende een opvolgerfunctie te construeren.

(W B) = (A A(A K)B) = (A B(A K B)) = (A B I) [32]

I was immers (A K Z) volgens formule [4a]. We zien dus dat 2 = (A B I) verkregen wordt uit 1 = I door op I de operator (A B) toe te passen. We zullen dit de successorfunctie S noemen.

S = (A B) [33]

Voor de volledigheid verifiëren we of door toepassing van S op 0 ook 1 ontstaat.

(S 0) = (A B(K I)) = I volgens formule [22]

Bewijs: (A B(K I)X Y) = (B X(K I X)Y) = (B X I Y) = (X(I Y)) = (X Y) =
= (I X Y)

We kunnen dus definiëren 3 = (S 2) enzovoort.

We zullen nu proberen de optelling te definiëren. Als M en N getaloperatoren zullen voorstellen, dan kunnen we zeggen dat de som van M en N bestaat uit het eerst N maal tellen (X) van een object (Y) en het daarna nog eens M maal tellen.

(+ M N X Y) = (M X(N X Y)) =
((M X)((N X)Y)) = (B(M X)(N X)Y) = (B B M X(N X)Y) =
(A(B B M)N X Y) = (B A(B B)M N X Y)

Daar M, N, X en Y willekeurig waren kunnen we volgens het extensionaliteitsbeginsel zeggen:

+ = (B A(B B)) [34]

Zo geeft

(+ 2 3 TEL SCHAAP) = (TEL(TEL(TEL(TEL(TEL SCHAAP)))))

en (+ 2 3) zal hetzelfde moeten geven als (S(S(S(S(S 0)))))!

Deze laatste verificaties zijn met de hand vrijwel niet uit te voeren. Alleen een machinaal afleidingssysteem garandeert een betrouwbare bewijsvoering; een mens maakt bij zulke ingewikkelde vormen met zoveel haakjes gauw fouten.

Voor de vermenigvuldigingsoperator kunnen wij een analoge redenering opzetten. Als (N TEL) het N maal tellen voorstelt, dan is (M(N TEL)) de operator die dat nog eens M maal doet. Dus
(* M N X Y) = (M(N X)Y) = (B M N X Y)
Daarmee zijn alle willekeurige objecten weer uitgefactoriseerd.
De conclusie is
* = B [35]
Verificatie op een computer geeft weer:
(B 2 3 X Y) = (X(X(X(X(X(X Y)))))) = (6 X Y)

Machtsverheffen kunnen we op de volgende manier afleiden:
(N^2 X Y) = (B N N X Y) = (N(N X)Y) = (2 N X Y)
(N^3 X Y) = (N(N(N X))Y) = (3 N X Y)
In het algemeen
(↑ N M X Y) = (M N X Y) = (C I N M X Y) waaruit we concluderen
↑ = (C I) [36]
De machtsverheffing is dus de verwisselingsoperator (C I), die we in formule [16] al gezien hebben. Alle arithmetische operatoren zijn steeds geschreven als prefixoperatoren, dus eigenlijk als functies met argumenten.

De methode, die we bij de afleiding van de machtsverheffing hebben gebruikt, hadden we ook voor de optelling kunnen gebruiken. We kunnen zeggen, dat N + 1 = (S N), N + 2 = (S(S N)) = (2 S N). Zo zal M + N de M'de successor van N zijn dus (M S N). Dan kunnen we dus schrijven:
(+ M N) = (M S N) = (C I S M N) Dus
+ = (C I S) [37]
Deze andere definitie van + blijkt niet gelijk te zijn aan de vorige, omdat hierbij van te voren verondersteld is dat N en M getallen zijn. Dat was bij de afleiding van + = (B A(B B)) niet het geval. Overigens werkt ook (C I S) toegepast op getaloperatoren goed en geeft dezelfde antwoorden.

Met behulp van de combinatorische logica kunnen we de verschillende eigenschappen van arithmetische operatoren bewijzen. Bijv. verloopt het bewijs van de associatieve eigenschap als volgt:
Zij M, N en P drie natuurlijke getallen. Dan moet gelden:
(N * (M * P)) = ((N * M) * P) in de gebruikelijke schrijfwijze. In prefixvorm dus (* N(* M P)) = (*(* N M)P)
Bewijs: (* N(* M P)X Y) = (B N(B M P)X Y) = (N(B M P X)Y) = (N(M(P X)Y)
en ook
(*(* N M)P X Y) = (B(B N M)P X Y) = (B N M(P X)Y) = (N(M(P X)Y)

Het is grappig dat men de combinatoren A, K, B, C, I en W kan uitdrukken in 0, +, *, en ↑ .

I = (0 0) [38]
C = (* * ↑ (* * ↑)(* * ↑)) [39]
K = (C 0) [40]
W = (C(*(+ I I)↑) [41]
A = (*(*(* W)C)(* *) [42]
B = * [35]

Het bewijs wordt weer aan de lezer overgelaten.

4. Lambdavormen

Tot nu toe kwamen in ons verhaal eigenlijk nog geen variabelen voor. Er was alleen sprake van willekeurige, maar bepaalde objecten. Het begrip variabele, waarvoor men een willekeurig object mag substitueren, heeft nog geen plaats gehad. Het begrip functie met formele parameters is gegeven door Church in een publicatie getiteld: The Calculi of Lambda Conversion. Daarin spelen de begrippen vrije en gebonden variabele een grote rol. De benadering vanaf de kant van de combinatorische logica heeft het voordeel, dat daarmee lambdavormen variabele-vrij gemaakt kunnen worden.

Als we in de wiskunde schrijven: $f(x) = x + y$
dan dient de variabele x alleen maar om de functie te beschrijven. We hadden net zo goed mogen schrijven: $f(t) = t + y$. Dit beschrijft dezelfde functie. Men noemt zo'n beschrijvingsvariabele een gebonden variabele. De y in de bovengenoemde functie is dan een vrije variabele. Men moet oppassen niet toevallig een vrije variabele voor een gebonden variabele te substitueren. De functie $f(y) = y + y$ beschrijft namelijk een geheel andere functie!

In de theorie van de programmeertalen noemt men de beschrijvingsvariabelen meestal formele parameters en de waarden, die men hiervoor substitueert de actuele parameters.

De letter f in het bovenstaande duidde alleen de naam van de functie aan. De gebonden variabelen behoren bij de beschrijving en niet bij de naam. We kunnen ook een functie beschrijven, zonder er een naam aan te geven. In navolging van Church zullen we hiervoor de lambdanotatie gebruiken (λx (vorm, die x al of niet als gebonden variabele bevat)).
In de programmeertaal LISP wordt voor de griekse letter het woord LAMBDA in deze vorm gebruikt. Ter wille van de kortheid en omdat computers in de regel geen griekse letters kennen, zullen we voor lambda de letter L gebruiken. We schrijven dus F = (L X(+ X Y))
In het linkerlid staat dus de naam van de functie F. Het rechterlid is de beschrijving van de functie, inclusief de parameterstructuur. De functie F toegepast op een actuele parameter N kunnen we schrijven als (F N), maar ook kunnen we schrijven (L X(+ X Y)N). De functie hoeft helemaal geen naam te hebben, maar kan direct als lambdavorm toegepast worden op de argumenten en geeft dan (+ N Y). In de literatuur wordt het vervangen van een formele variabele in de beschrijving door een andere wel alpha-reductie en het toepassen van een lambdavorm op een actuele parameter wel beta-reductie genoemd.

Functies van meer variabelen kunnen we altijd opvatten als het achtereenvolgens toepassen van functies van één variabele op steeds één actuele variabele tegelijk. Bijv.
(L X(L Y(vorm die X en Y als gebonden var kan bevatten))P Q)
We kunnen dan zeggen, dat eerst (L X(binnenvorm)) op P werkt en het resultaat dat zelf weer begint met (L Y ...) daarna op Q werkt.

We zullen nu de lambdacalculus formeel definiëren vanuit de combinatorische logica met drie definities.

(L X X) = I [43]

(L X Y) = (K Y) met Y ≠ X [44]

(L X(P Q)) = (A(L X P)(L X Q)) [45]

De eerste regel zegt iets over een enkelvoudige variabele, die de gebonden variabele is. Toegepast op een willekeurig object N levert dit dus altijd N zelf op. (L X X N) = N = (I N).

De tweede regel zegt iets over een enkelvoudige variabele die niet gelijk is aan de gebonden variabele. Toegepast op een willekeurig object N is er in de vorm geen X waarvoor N gesubstitueerd kan worden. Het resultaat is dus alleen Y. (L X Y N) = (K Y N) = Y

De derde regel is de compositieregel voor vormen, die niet enkelvoudig zijn. Men kan op deze manier altijd een vorm tot atomen afbreken en de eerste twee regels toepassen.

Ook deze keuze van definitie is aannemelijk als we bedenken dat als P en Q zelf X bevatten, dus functies van X zijn, de functie (P Q) toegepast op een actueel object N moet geven: (P N(Q N)), namelijk N gesubstitueerd voor de parameter X in P en dat toegepast op N gesubstitueerd in Q. Maar dat is precies wat we krijgen door (A(L X P)(L X Q) toe te passen op N
(A (L X P)(L X Q)N) = (L X P N(L X Q N)) = (P N(Q N))

De hiernavolgende afgeleide regels kunnen soms zeer van pas komen bij herleidingen:
(L X(F X)) = (A(L X F)(L X X)) = (A(K F)I) = F [46]
het laatste ingevolge de reeds eerder afgeleide formule [18]. Dit is natuurlijk in volkomen overeenstemming met (L X(F X)N) = (F N).

Formule [45] kan ook als volgt herleid worden:
(L X(P Q)) = (A(L X P)(L X Q)) = (A(K P)(K Q)) = (K(P Q)) [47]
het laatste ingevolge formule [20] . We hadden dit ook direct kunnen afleiden door (P Q) te beschouwen als een nieuw object en daarna formule [44] toe te passen.

Een andere nuttige formule bij herleidingen met de hand is:
(L X(F X G)) = (A(L X(F X))(L X G)) = (A F(K G)) [48]
We merken direct de overeenkomst op met formule [15]. We hadden immers ook kunnen afleiden:
(L X(F X G)) = (L X(C F G X)) = (C F G) = (A F(K G))

Alle vormen, die we met combinatoren kunnen schrijven, kunnen ook als lambdavorm geschreven worden. Zo kunnen we zeggen dat A beschouwd kan worden als functie van drie variabelen:
A = (L X(L Y(L Z(X Z(Y Z))))) [49]
Op dergelijke wijze is
K = (L X(L Y X)) [50]

Op deze wijze kunnen we ook snel formules vinden voor bijv. de verwisselingsoperator.

(L X(L Y(Y X))) = (L X(A I(K X)) = (A(L X(A I))(L X(K X))) = (A(K(A I))K)

en ook

(C I) $\underset{14}{=}$ (B(A I)K) $\underset{7}{=}$ (A(K(A I))K)

Het is niet altijd het handigste om de lambdaregels toe te passen. Soms kunnen we met gebruikmaking van de andere operatoren B, C, W en een beetje slimheid kortere vormen vinden, maar voor machinale afleiding zijn de regels zeer geschikt. We zullen deze andere methode demonstreren aan een zeer merkwaardige operator:

D = (L X(L Y(L N(N(K Y)X)))) [51]

Deze operator, toegepast op drie argumenten X, Y en N, waarvan de laatste een getal moet zijn, heeft de volgende werking:

<u>procedure</u> D(x,y,n); <u>if</u> n = 0 <u>then</u> x <u>else</u> y;

We zullen dit de decisiefunctie noemen. Deze vorm is voor het eerst gevonden door Bernays. We verifiëren even voor de gevallen 0 en 1.

(0(K Y)X) = (K I(K Y)X) = (I X) = X

(1(K Y)X) = (I(K Y)X) = (K Y X) = Y

We kunnen gemakkelijk verifiëren, dat ook voor getallen > 1 de uitkomst Y is. Merk hierbij op, dat het niet gekozen argument door de K weggegooid wordt nog voordat het ooit geëvalueerd is! Dit is namelijk het essentiële van een decisieoperator zoals <u>if</u>. Zou hij immers wel beide argumenten eerst berekenen, dan zou hiermee een recursieve definitie van een functie onmogelijk worden.

We zullen nu een gesloten vorm voor D op de volgende manier afleiden. We gaan trachten de variabelen N, Y en X, die in de verkeerde volgorde staan en bovendien nog gecombineerd voorkomen, achteraan te krijgen. In feite hebben we deze methode al eerder toegepast bij de formule [34].

(N(K Y)X) = (B N K Y X) =	haakjes weggewerkt met B
(C B K N Y X) =	N naar achteren met C
(C(C B K)Y N X) =	N verder naar achteren
(C(C(C B K)Y)X N) =	N op zijn plaats gebracht
(B C(C(C B K))Y X N) =	Y buiten de haakjes
(C(B C(C(C B K)))X Y N)	X en Y in de goede volgorde

Hieruit concluderen we tot

D = (C(B C(C(C B K)))) [52]

= (A(K(A(A(K(A(A(K(A(K A)K))A)(K K)))(A(K(A(A(A(K A)K)(K K))))K))))))K)

dit laatste voor degene, die er persé op staat alles in A en K uit te drukken. Het zal duidelijk zijn, dat dit eigenlijk alleen foutloos door een machine kan gebeuren. Deze combinatorvorm van D geeft een parametervrije uitdrukking voor de decisie. Het grote voordeel is dat deze vorm automatisch geprotegeerd is tegen het ongeoorloofd alpha-substitueren van een vrije variabele voor een gebonden variabele.

We kunnen op een bijzonder overtuigende wijze de werking van de decisiefunctie D demonstreren. Beschouwen we de vorm (W W W), dan zien we dat de ontwikkeling naar de W-regel weer (W W W) geeft. Het in ons geval geheel recursief geschreven programma in LISP om dergelijke vormen te berekenen geeft op zo'n oneindige ontwikkeling de reactie: sorry, ruimte niet voldoende. De decisiefunctie D toegepast op

(D X(W W W)0) = X

en komt zoals het hoort nooit aan (W W W) toe. Ook geeft

(D(W W W)Y 3) = Y

Het is soms handiger om een kleine variant van de decisiefunctie te gebruiken

E = (L N(L X(L Y(N(K X)Y))))

= (C B K) = (A B(K K)) [53]

met de werking:

<u>procedure</u> E(n,x,y); <u>if</u> n ≠ 0 <u>then</u> x <u>else</u> y

5. Recursieve functies

De decisiefuncties geven ons het hulpmiddel om recursieve functies te definiëren. Daarvoor hebben we wel de voorganger- of predecessorfunctie nodig, die gedefiniëerd is door:

(P 0) = 0 en

(P(S N)) = N waarin N een getal [54]

We kunnen nu P definiëren met een hulpfunctie

G = (L F(D(S(F O))(F O)))

= (B(W(B D S))(C(C B K)I)) = (B(A(B D S)I)(A(A B(K K))(K I))) [55]

P = (L N(N G(K O)1))

= (C(C(C I G)(K O))I)

= (A(A(A I(K G))(K(K(K I))))(K I)) [56]

Voor het bewijs, zie Stenlund. Volledig uitgeschreven luidt P:

P = (A(A(A(A K K)(K(A(K(A(A(K(A(K(A(A(K(A(A(K(A(K A)K))A)(K K)))
(A(K(A(A(A(K A)K)(K K)))K))))K))(A(A(K A)K)))(A K K)))
(A(A(A(K A)K)(K K))(A K)))))(K(K(A K))))(A K)) [57]

Zo zal dus zijn:

(P 1 X Y) = Y

(P 2 X Y) = (X Y)

(P 3 X Y) = (X(X Y)) enz.

De begrensde aftrekking uit de theorie van de recursieve functies kan nu ook gemakkelijk neergeschreven worden. Waar (P N) = N∸1 en (P(P N)) = N∸2 kan men in het algemeen schrijven (M P N) = N∸M dus: (MINUS N M) = (M P N) = (C I P M N) = (C(C I P)N M). Dit geeft:

MINUS = (C(C I P)) = (C(A I(K P))) [57a]

Met behulp van P kunnen wij recursieve functies definiëren. Bijv. kunnen wij in ALGOL 60 schrijven voor de faculteitsfunctie:

<u>integer</u> <u>procedure</u> FAC(N); FAC := <u>if</u> N ≠ 0 <u>then</u> N * FAC(N-1) <u>else</u> 1;

Hetzelfde geschreven in een lambdavorm met combinatoren:

FAC = (L N(E N(B N(FAC(P N)))I)) [58]

We zullen de vorm parametervrij maken als volgt:

(E N(B N(FAC(P N)))I) =

= (E N(B N(B FAC P N))I)

= (E N(A B(B FAC P)N)I)

= (A E(A B(B FAC P))N I)

= (C(A E(A B(B FAC P)))I N)

= (A(A E(A B(B FAC P)))(K I)N)

Daarmee is tenslotte

FAC = (A(A E(A B(B FAC P)))(K I)) [58a]

Deze recursieve vorm van definitie is voor een machine wel degelijk werkzaam als we maar niet de fout begaan om in het rechterlid voor FAC weer de definitie van FAC te substitueren. Zonder argumenten komen we dan in de oneindige recursie terecht. In de volgende paragrafen zullen we zien hoe ook dit nog verholpen kan worden.

Reeds nu hebben we met de combinatoren alle mogelijkheden die de theorie van de primitieve recursieve functies ons biedt. De K is gelijkwaardig met het schema van de constantheid. De I is de identiteit. K kan ook gebruikt worden voor de projectie. Functionele compositie kan verkregen worden met B en A. De successorfunctie is aanwezig en de recursie kan uitgedrukt worden met de voorgangerfunctie. Weliswaar hebben we nog geen gesloten vorm verkregen voor het schema van de primitieve recursie, maar dat zal ons in de volgende paragraaf ook lukken. Rest nog het schema van de minimalisering: gevraagd een functie (F Y) die voor alle Y de minimale X geeft waarvoor (G X Y) = O. Ook dit kan geheel in combinatoren worden uitgedrukt. Voor een behandeling verwijzen we naar Stenlund of Rosenbloom. Hiermee is aangetoond dat de theorie van de combinatoren gelijkmachtig is met de theorie van de algemene recursieve functies.

6. De Fixpointoperator

Wij maken even een schijnbare zijsprong en gaan de theorie van de combinatoren toepassen op een probleem uit de logica, nl. de paradox van Russell. In een bepaalde vorm gegoten kan men deze paradox als volgt formuleren:

> Bevat de verzameling van alle verzamelingen, die zichzelf niet als element bevatten, zichzelf als element of niet?

Geeft men hierop een bevestigend antwoord, dan is dit een verzameling die zichzelf wel bevat en dat mag niet. Geeft men een ontkennend antwoord, dan had hij juist wel in de verzameling thuisgehoord. We kunnen dit zo in formule brengen:

F = (L X(N(X X))) [59]

Daarin stelt N de negatie voor. F is dus een functie, zodanig dat geldt N(X X), d.w.z. dat hij niet op zichzelf mag worden toegepast. Immers substitueren we F voor X, dan krijgen we:

(F F) = (N(F F)) [60]

of: FF is NOT FF, hetgeen een tegenspraak inhoudt.

We zullen nu F als combinator ontwikkelen.

F = (L X(N(X X))) =

(L X(B N X X)) =

(L X(W(B N)X)) =

(W(B N)) = (B W B N)

Het object (F F) hield een tegenspraak in. De ontwikkeling van (F F) is:

(F F) = (B W B N(B W B N)) =

(A(B W B)(B W B)N) =

(W A(B W B)N)

Daar de eigenschappen van N als combinator eigenlijk in het geheel niet ter sprake zijn gekomen, kunnen we hiervoor ook elk ander object gebruiken. De combinator (W A (B W B)) maakt dus van een willekeurig object een paradoxaal object. Curry noemde deze combinator daarom de paradoxale combinator. Wij zullen hem om zo dadelijk uiteen te zetten redenen de fixpointoperator noemen en aanduiden met R

R = (W A(B W B)) [61]

We beschouwen hierna een probleem dat op het eerste gezicht niets met het vorige heeft te maken. Een procedure (of functie, wat in ALGOL bijna hetzelfde is) wordt gedefinieerd door een declaratie van de vorm:

<u>procedure</u> Q = < een of andere routine text >

Deze routinetext kan bij zijn beschrijving zelf weer de procedure P gebruiken en is in dat geval een functie van Q zelf. Zo'n procedure noemen we recursief. Laten we deze functie F noemen, dan kunnen we schrijven:

<u>procedure</u> Q = F(Q)

of nog korter, met gebruikmaking van de iets zorgelozer mathematische schrijfwijze van een definitie en de combinatorische schrijfwijze van functies:

Q = (F Q) [62]

We zien dus dat de functie F, toegepast op Q, Q zelf weer oplevert. Daarom wordt Q een dekpunt of fixpoint van F genoemd. We kunnen ons afvragen of we bij elke gegeven functie F zo'n dekpunt kunnen vinden. Het antwoord luidt bevestigend en is zeer verrassend. De in de vorige paragraaf

gevonden paradoxale combinator R blijkt tevens deze fixpointoperator te zijn, waarmee het dekpunt van F gevonden kan worden door:

Q = (R F) [63]

Bewijs:

Q = (R F) =
(W A(B W B)F) =
(B W B F(B W B F)) = [64]
(W(B F)(B W B F)) =
(B F(B W B F)(B W B F)) =
(F(B W B F(B W B F))) =
(F Q) ingevolge formule [64]

We kunnen dus R als recursieoperator gebruiken. Zomaar toegepast op een ongespecificeerd object X kan de ontwikkeling nooit klaarkomen. Immers (R X) = (X(R X)) = (X(X(R X))) = (X(X(X(R X)))) ... enz.

Maar in een concreet geval van een functie werkt R heel normaal.

Beschouwen we nogmaals formule [58a], dan kunnen we verder herleiden:

FAC = (A(A E(A B(B FAC P)))(K I)) =
(C A(K I)(A E(A B(B FAC P)))) =
(A A(K O)(A E(A B(B FAC P)))) =
(B(A A(K O))(A E)(A B(B FAC P))) =
(B(B(A A(K O))(A E))(A B)(B FAC P)) =
(B(B(B(AA(K O))(A E))(A B))(B FAC)P) =
(B(B(B(B(AA(K O))(A E))(A B))B)FAC P) =
(C(B(B(B(B(AA(K O))(A E))(A B))B)P FAC)

Wij hebben hiermee FAC uitgefactoriseerd en kunnen schrijven

! = (C(B(B(B(B(AA(K O))(A E))(A B))B)P) en [65]
FAC = (R !)

(R ! 3 X Y) geeft heel normaal (X(X(X(X(X(X Y))))))). Mathematisch kan men wonderlijke resultaten afleiden zoals het volgende:

QUEER = (R K)

Deze operator zou de eigenschap hebben, dat hij toegepast op willekeurige X zichzelf oplevert. Immers:

(QUEER X) = (R K X) = (K QUEER X) = QUEER

Probeert men dit echter uit te voeren op een computer, dan loopt hij vast door gebrek aan ruimte om (R K) te expanderen. Het zal waarschijnlijk wel

een undecidable problem zijn om uit te maken wanneer (R VORM) in constructieve zin, dus in eindige tijd een oplossing zal geven.

7. Andere getalrepresentaties

De in hoofdstuk 3 geschetste wijze van getalvoorstelling, die van Church afkomstig is, is niet de enig mogelijke. In Curry, Hindley en Seldin komt een aanduiding voor van een getalrepresentatie van Scott die echter niet nader uitgewerkt is en nauwelijks een zinvolle visualisering toelaat. Daarom zal ik hier nog een nieuw systeem presenteren, dat ik onlangs heb uitgewerkt. Vele operatoren zijn in dit systeem veel eenvoudiger en vele resultaten met de fixpointoperator en recursieve schema's konden eerst praktisch onderzocht worden dank zij dit systeem.

In het systeem van Church stelden we een getal voor door een handeling X, die N maal op Y werd toegepast. In het nieuwe systeem, dat we voor het gemak P-systeem zullen noemen, wordt N toegepast op X en Y voorgesteld door

($\underline{N}$ X Y) = (X Y Y ... Y) [66]

Ter onderscheiding van getallen in het Church systeem zullen we de getallen in het P-systeem voorstellen door onderstreping.

($\underline{0}$ X Y) moet dan leveren X. Dat gebeurt juist met de K, dus

$\underline{0}$ = K [67]

($\underline{1}$ X Y) moet leveren (X Y). Dus $\underline{1}$ is weer I, net als bij Church

$\underline{1}$ = I [68]

($\underline{2}$ X Y) moet leveren (X Y Y). Dat gebeurt juist met de W, dus

$\underline{2}$ = W [69]

Het vinden van de juiste combinatoren voor de successor en de optelling is een moeizaam proces geweest, dat zonder hulp van de machine niet mogelijk zou zijn geweest. Achteraf kan men de resultaten heel gemakkelijk terugvinden.

Beginnen we met de optelling, dan zien we dat de operator B toegepast op $\underline{N}$, $\underline{M}$, X en Y het volgende geeft. ($\underline{N}$ en $\underline{M}$ zijn getallen en X en Y zijn willekeurige objecten.)

$$(B\ \underline{N}\ \underline{M}\ X\ Y) = (\underline{N}(\underline{M}\ X)Y) = ((\underline{M}\ X)\underbrace{Y\ \ldots\ Y)}_{N\text{ stuks}}$$

$$= (\underline{M}\ X\ Y\ \underbrace{Y\ \ldots\ Y)}_{N-1\text{ stuks}} = \qquad [70]$$

$$= (X\ \underbrace{Y\ \ldots\ Y}_{M\text{ stuks}}\ \underbrace{Y\ \ldots\ Y}_{N-1\text{ stuks}})$$

We zien dat B de operatie N+M-1 uitvoert. Bij de expansie van M wordt de eerste Y van de expansie van N geleend. Dit gaat natuurlijk alleen goed als $\underline{N} > 0$ is.

De successorfunctie $\underline{S}$ volgt direct uit B door voor $\underline{N}$ te nemen $\underline{2}$ = W

$\underline{S}$ = (B W) [71]

Immers krijgen we dan 2+M-1 = M+1

Het is illustratief om ook de lambda-vorm van $\underline{S}$ op te zoeken. Dat is het omgekeerde proces van het vinden van de combinator uit de lambda-vorm.

($\underline{S}$ $\underline{N}$ X Y) = (B W $\underline{N}$ X Y) =

= (W($\underline{N}$ X)Y)

= ($\underline{N}$ X Y Y)

Hieruit vinden we:

$\underline{S}$ = (L $\underline{N}$(L X(L Y($\underline{N}$ X Y Y)))) [71a]

Nu gaf ($\underline{N}$ X Y) een X met N Y's erachter. De successor voegt daar nog een extra Y aan toe.

Met behulp van de successor vinden we ook de som door te nemen:

($\underline{+}$ $\underline{N}$ $\underline{M}$) = (B($\underline{S}$ $\underline{N}$)$\underline{M}$) = (B B(B W)$\underline{N}$ $\underline{M}$) dus

$\underline{+}$ = (B B(B W)) = (B B $\underline{S}$) [72]

Ook hier krijgen we een goed inzicht in de werking door de lambda-vorm te ontwikkelen:

($\underline{+}$ $\underline{N}$ $\underline{M}$ X Y) = (B B(B W)$\underline{N}$ $\underline{M}$ X Y)

= (B(B W $\underline{N}$)$\underline{M}$ X Y)

= (B W $\underline{N}$($\underline{M}$ X)Y)

= (W($\underline{N}$($\underline{M}$ X)Y))

= ($\underline{N}$($\underline{M}$ X)Y Y) Hieruit vinden we:

$\underline{+}$ = (L $\underline{N}$(L $\underline{M}$(L X(L Y($\underline{N}$($\underline{M}$ X)Y Y))))) [72a]

waaruit we weer zien, dat het resultaat hetzelfde is als bij formule [70] met nog een extra Y toegevoegd.

Voor de meeste recursieve functies hebben we helemaal niet de volledige predecessorfunctie nodig, waarvoor geldt, dat (P O) = O omdat dat geval door de decisiefunctie toch al vermeden wordt. Een eenvoudiger predecessorfunctie krijgen we door in (B N M) voor M te nemen O = K.
We krijgen dan N + O - 1 = N - 1. (N mocht niet O zijn !) Dus:
(P N) = (B N K) = (C B K N) Hieruit volgt
P = (C B K) = (A B(K K)) [73]
Het is frappant dat de predecessor in het P-systeem dezelfde operator is als de decisiefunctie E in het gewone systeem, maar we kunnen er geen conclusies aan verbinden.

De lambda-vorm laat zich weer als volgt herleiden:
(C B K N X Y) = (B N K X Y) = (N(K X)Y) waaruit we besluiten tot:
P = (L N(L X(L Y(N(K X)Y)))) [73a]
Ook hier is de werking duidelijk. (N(K X)Y) produceert (K X Y ... Y) met N Y's, maar de K annihileert daarna de eerste Y, zodat er dan nog precies N-1 stuks zijn. We kunnen opmerken dat de lambda-vorm voor P nog een combinator bevat. We hadden die nog wel weg kunnen werken door voor K te substitueren (L F(L G F)) volgens formule [50] waardoor wordt verkregen P = (L N(L X(L Y(N (L G X)Y)))). Hierin komt een variabele voor een nieuwe-lambda voor. Men noemt dit in tegenstelling tot de lambda-vormen die we tot nu toe gezien hebben een niet-normaalvorm. Het heeft uit toepasbaarheidsoogpunt geen enkel belang of een lambda-vorm op normaalvorm gebracht kan worden of niet. Als constructivist zullen we deze onderscheiding aan de theoretici overlaten.

De laatste bouwsteen, die nog nodig is, is de decisiefunctie E.
Deze blijkt gerealiseerd te worden door:
E = (L N(L X(L Y(N K I I(K X)Y)))) [74]
= (A(A(A(B(B(B(A B(K K)))))K)(K K))(K I))(K I))
De omwerking van de lambda-vorm op de zuivere combinatorvorm wordt aan de lezer overgelaten.

We zullen deze combinatoren tenslotte nog gebruiken voor een aantal voorbeelden. Om te beginnen de vermenigvuldiging. Het is helaas niet gelukt om hiervoor een zo eenvoudige gesloten uitdrukking te vinden als in het Churchsysteem. Daarom definiëren we recursief:

integer procedure MAAL(n,m) ; MAAL := if n ≠ 0 then m+m*(n-1) else 0;

of in combinatoren:

* = (L N(L M(E N(B B S M(* M (P N)))O) [75]

Dit geeft na vlijtig herleiden:

* = (B(A A(K(K K)))(A(B B E)(B(A(B B S))(B * P)))) [75a]

Door parametrisering op * kan men een recursievrije uitdrukking vinden voor *

MAAL = (L X(B(A A(K(K K)))(A(B B E)(B(A(B B S))(B X P)))) [76]

De recursieoperator R geeft dan de vermenigvuldiging bijv.

(R MAAL 3 3 X Y) = (X Y Y Y Y Y Y Y Y Y)

Een kwadraatfunctie kan men maken met de recursie:

f(n) = if n ≠ 0 then f(n-1) + 2*n - 1 else 0

In combinatoren uitgedrukt geeft 2*n - 1 (B N N) = (W B N). Het geheel

F = (L N(E N(+ (W B N)(F(P N)))K)) [77]

= (A(A E(A(B +(W B))(B F P))(K K))

Parametrisering op F geeft weer een recursievrije vorm.

SQ = (L X(A(A E(A(B +(W B))(B X P)))(K K)) = [78]

= (A(A(K A)(A(K(A(A(A(A(A(K(A(K(A(K(A(A
(K A)K)(K K)))))))))(K K))(A K))(A K))))
(A(K(A(A(K(A(K(A(K A)K))(A(K(A A(A K)))
)))(A(A(K A)K)(A K K)))))(A(A(K A)K)(K(
A(A(K A)K)(K K)))))))(K(K K)))

Zoals te verwachten geeft

(F SQ 3 X Y) = (X Y Y Y Y Y Y Y Y Y)

Op dergelijke manier kan men de getallen van Pascal berekenen met

PASCAL = (L X(L N(E N(+(X(P N))N)O))) =

= (L X(A(A E(W(B +(B X P))))(K K))) [79]

Dit geeft bijv. het resultaat:

(R PASCAL 3 X Y) = (X Y Y Y Y Y Y)

Er is voor de getallen in het P-systeem een heel aanschouwelijke betekenis te geven aan een representatie door voor X te kiezen ZERO en voor Y te kiezen PLUSONE. Men krijgt dan voor

($\underline{\text{N}}$ ZERO PLUSONE) = (ZERO $\underbrace{\text{PLUSONE PLUSONE ... PLUSONE}}_{\text{N stuks}}$)

Er is een gemakkelijke overgang van getallen in het Church-systeem op getallen in het P-systeem door op te merken, dat $\underline{\text{N}}$ de N^{de} successor van K is.

$\underline{\text{N}}$ = (N (B W) K) [80]

De omgekeerde overgang laat zich ook uitvoeren met een functie die we even U zullen noemen en die dus de eigenschap moet hebben dat

N = (U $\underline{\text{N}}$) [81]

Deze functie laat zich recursief als volgt uitdrukken:

U = $\underline{\text{if}}$ $\underline{\text{N}}$ $\neq$ $\underline{\text{0}}$ $\underline{\text{then}}$ S(U($\underline{\text{P}}$ $\underline{\text{N}}$) $\underline{\text{else}}$ (K I), of in combinatoren:

U = (L $\underline{\text{N}}$($\underline{\text{E}}$ $\underline{\text{N}}$(S(U($\underline{\text{P}}$ $\underline{\text{N}}$)))(K I))) =

(A(A $\underline{\text{E}}$(B S(B U $\underline{\text{P}}$)))(K(K I))) [82]

Desgewenst kan door parametrisering op U deze formule recursievrij gemaakt worden. Het probleem van de aftrekking in het P-systeem is daarmee ook opgelost op analoge manier als in formule [57a]. We vinden:

($\underline{\text{MINUS}}$ $\underline{\text{N}}$ $\underline{\text{M}}$) = (M $\underline{\text{P}}$ $\underline{\text{N}}$) = (U $\underline{\text{M}}$ $\underline{\text{P}}$ $\underline{\text{N}}$) = (C U $\underline{\text{P}}$ $\underline{\text{M}}$ $\underline{\text{N}}$)== (C(C U $\underline{\text{P}}$)$\underline{\text{N}}$ $\underline{\text{M}}$) dus

$\underline{\text{MINUS}}$ = (C(C U $\underline{\text{P}}$)) [83]

8. Nog onopgeloste problemen

Het is misschien nuttig om enkele op dit moment nog onopgeloste problemen te vermelden (4 juli 1973).

a) Een eenvoudige predecessor in het Church-systeem waarvoor alleen geldt (P(S N)) = N en niet (P 0) = 0

b) Een directe vermenigvuldigoperator in het P-systeem.

NOTATIES EN DEFINITIES

A	= (L X(L Y(L Z(X Z(Y Z)))))	Associator
B	= (L X(A(K X))) = (A(K A)K)	Haakjeszetter
C	= (L X(B(A X)K)) = (A(B B A)(K K))	Verwisselaar
D	= (L X(L Y(L N(N(K Y)X)))) = (C(B C(C(C B K))))	Decisiefunctie
E	= (L N(L X(L Y(N(K X)Y)))) = (C B K)	Eenv.decisiefunctie
E̲	= (L N(L X(L Y(N K I I(K X)Y))))	Decisie in P-systeem
F	Algemene notatie voor functie	
G	Algemene notatie voor functie	
I	= (L X X) = (A K Z)	Identiteit
K	= (L X(L Y X))	Konstantsheidfunctie
L	Lambda, altijd gevolgd door gebonden variabele	
M	Getallen	
N	Getallen	
O	= (K I)	Nul
P	= (C(C(C I G)(K O))I)	Predecessor
P̲	= (C B K)	Predecessor in P-syst.
Q	Alg.variabele	
R	= (W A(B W B))	Recursie of fixpoint
S	= (A B)	Successor
S̲	= (B W)	Successor in P-systeem
W	= (L X(A X I)) = (A A(A K)) = (C A I)	Dupliceringsoperator
X	Onbepaald object of gebonden variabele	
Y	" " " " "	
Z	" " " " "	
+	= (B A(B B))	Optelling
+̲	= (B B(B W))	Optelling in P-systeem
*	= B	Vermenigvuldiging

Referenties

Curry, H.B., Feys, R. and Craig, W., *Combinatory logic, Vol.1,* Studies in logic and the foundations of mathematics, North-Holland Publ.Co., Amsterdam, (1958).

Curry, H.B., Hindley, J.R. and Seldin, J.P., *Combinatory logic, Vol.2,* Studies in logic and the foundations of mathematics, 65, North-Holland Publ.Co., Amsterdam, (1972).

Rosenbloom, P., *The elements of mathematical logic,* Dover Publ., New York, (1950).

Stenlund, S., *Combinators, lambda-terms and proof theory,* Reidel Publ.Co., Dordrecht, (1972).

Barendregt, H.P., *Some extensional term models for combinatory logics and lambda-calculi,* thesis, R.U.Utrecht, Utrecht, (1971).

BETROUWBAARHEID VAN PROGRAMMA'S

door

E.W. Dijkstra

Mathematische uitspraken plegen zich door drie eigenschappen te onderscheiden:

a) ze zijn algemeen in de zin, dat ze op een groot aantal gevallen van toepassing zijn; (men formuleert een eigenschap van "alle niet ontaarde driehoeken"),

b) ze zijn heel precies; (dit in tegenstelling tot alle uitspraken, die hun algemeenheid aan vaagheid ontlenen),

c) er bestaat een bewijstraditie, waardoor de kans op onjuistheid van de uitspraak drastisch gereduceerd kan worden; (hierbij tekenen we aan, dat er voor het bereiken van een dergelijk betrouwbaarheidsniveau geen alternatieven bekend zijn).

Zodra wij rekenautomaten aan het werk zetten met de bedoeling aan de uitkomsten consequenties te verbinden, zijn wij daartoe slechts gemachtigd voorzover wij ons geloof kunnen rechtvaardigen, dat de geproduceerde uitkomsten inderdaad het antwoord op het door ons gestelde probleem zijn. Een eerste vereiste daartoe is, dat wij ons ervan kunnen overtuigen dat het programma correct is. Hoe kunnen wij deze overtuiging bereiken?

In de begintijd heeft men dit door "programma testen" geprobeerd te bereiken: men onderwierp het programma aan een aantal bekende testgevallen, proefberekeningen, waarvan het antwoord bekend was: als in deze testgevallen het juiste resultaat bereikt werd, nam men aan, dat het programma in alle gevallen het correcte resultaat zou produceren. De bittere ervaring heeft geleerd, dat deze laatste aanname ongerechtvaardigd is.

Het punt is, dat de klasse mogelijk bedoelde rekenprocessen, die onder controle van een niet helemaal triviaal programma uitgevoerd moeten worden, zo enorm groot is, dat men met de testgevallen slechts een absoluut verwaarloosbare fractie metterdaad kan proberen en hele subklassen van in een of andere zin "kritische gevallen" kan en zal missen. Sterker: de klasse van de mogelijke berekeningen is een klasse van zo sterk gestructureerde objecten, dat, wanneer van de 1000 testgevallen er 1 is misgegaan, de verwachting dat ook in de toekomst van de 1000 testgevallen er gemiddeld 1 zal misgaan helemaal ongerechtvaardigd is: de berekeningen zijn onvoldoende gelijksoortig om hier zinvol statistiek op te kunnen bedrijven. (Dit is iets, wat allerlei mensen maar heel moeilijk kunnen geloven.) Om het vereiste betrouwbaarheidsniveau te halen, rest ons slechts één ding, nl. bewijzen, dat het programma correct is.

Om dit te kunnen doen moet aan drie voorwaarden voldaan zijn:

a) de vereiste eigenschappen moeten voldoende scherp en hanteerbaar geformuleerd zijn (anders hoeven we over "correctheid" helemaal niet te praten),

b) er moet een bewijstraditie zijn (inclusief stellingen, axioma's en beproefde redeneerpatronen),

c) de voor een correctheidsbewijs benodigde hoeveelheid "manipulatie" moet binnen de grenzen van het mogelijke blijven.

De eerste pogingen om correctheidsbewijzen te leveren waren niet bemoedigend. In retrospectie is dat ook heel begrijpelijk.

Om van een programma de correctheid te bewijzen zal men een formele definitie van de semantiek van de programmeertaal, waarin het programma is uitgedrukt, moeten hebben en de eerste pogingen om zulke formele definities te geven hebben, hoewel tot ondubbelzinnige definities - en dat was in die tijd al heel wat! -, niet geleid tot axiomastelsels, die zich soepel leenden tot basis van correctheidsbewijzen. In dit opzicht lijken de tijden inmiddels veranderd.

Een tweede oorzaak van de aanvankelijk ontmoedigende ervaringen was gelegen in het feit, dat men zich in zijn optimisme stortte op het leveren van correctheidsbewijzen van programma's, geschreven in toen gangbare programmeertalen, die meestal niet vrij waren van anomalieën. Een zekere barok-

heid kon hieraan niet ontzegd worden en ze waren zeker niet ontworpen in een tijd, dat men zich over bewijsbaarheid van correctheid al veel zorgen maakte.

De grootste doorbraak is echter gekomen door de ontdekking, dat de hoeveelheid vereiste manipulatie sterk kon afhangen van de structuur van het programma en dat een van de dingen, die men met structurering van een programma kon nastreven, juist vermindering van de bewijslast was. Een onmiddellijk gevolg van deze ontdekking is geweest, dat men het correctheidsprobleem constructiever ging benaderen. In plaats van eerst een programma te schrijven en dan te proberen om te bewijzen dat het goed was, ging men correctheidsbewijs en programma hand in hand ontwikkelen; op het moment, dat het correctheidsbewijs vorm kreeg, schreef men een programma, waarop dit correctheidsbewijs van toepassing was. Een extra voordeel van deze benaderingswijze was, dat op dat moment de correctheidsoverwegingen zich ontpopten tot een vruchtbaar heuristisch hulpmiddel.

Een van de machtigste hulpmiddelen is de invariantiestelling voor loops van C.A.R. Hoare:

Indien voor de statement S de voorwaarde P <u>and</u> B een voldoende pre-conditie is om te garanderen dat, mits de uitvoering van S eindigt, aan de post-conditie P dan voldaan zal zijn, is voor de statement

<u>while</u> B <u>do</u> S <u>od</u>

de pre-conditie P voldoende om in geval van beëindiging de post-conditie P <u>and</u> <u>non</u> B te garanderen.

Het belang van deze stelling is daarin gelegen, dat hij zich zeer wel blijkt te lenen voor de constructie van programma's. Om een bepaalde relatie R te bewerkstelligen, kiest men een P en een B , zodat P <u>and</u> <u>non</u> B de eindrelatie R impliceert. Het programma krijgt dan de algemene vorm

vestig de geldigheid van P;
<u>while</u> B <u>do</u> onder invariantie van P een "stap"
in de richting van "<u>non</u> B" <u>od</u>

De stelling is in het bijzonder machtig, omdat wij ons bij de bewijsvoering er niet over hoeven uit te laten, hoe effectief de stap in de richting "<u>non</u> B" nu wel precies is: zolang we ons ervan kunnen overtuigen dat de toestand "<u>non</u> B" in een eindig aantal stappen bereikt zal worden,

is dit stuk van het correctheidsbewijs compleet. Doordat de stelling van Hoare zich er niet over uitspreekt, hoe vaak de herhaalbare statement herhaald wordt, hebben we hier een stuk gereedschap dat ons in staat stelt programma's met een deterministisch netto effect te construeren onder gebruikmaking van het semi-deterministische primitivum "een stap in de goede richting". Het is dit gebruik van semi-deterministische primitiva, dat onder de naam "strategische abstractie" bezig is burgerrecht te verkrijgen.

Bijna alle mij bekende programma's om bij een gegeven puntenwolk in het platte vlak het convex omhulsel te vinden zijn via strategische abstractie bijvoorbeeld afbeeldbaar op hetzelfde abstracte programma van de volgende vorm, waarin de mogelijke waarden van de variabele "convex omhulsel" het convex omhulsel van een niet-lege deelverzameling van de gegeven punten zijn.

```
initialiseer het convex omhulsel, zodat het een of meer
punten omvat;
while er bestaat een punt buiten het convex omhulsel do
     kies een punt buiten het convex omhulsel en noem dat q;
     pas het convex omhulsel aan, zodat ook punt q omvat is od
```

In hun uiteindelijke uitwerking kunnen de hierop geënte programma's nog enorm verschillen, bv. doordat ze zich in de keuze van het punt q altijd zullen beperken tot een punt dat op het uiteindelijke contour zal blijken te liggen of doordat ze zich deze beperking niet opleggen.

Een tweede techniek begint burgerrecht te verkrijgen onder de naam "representationele abstractie". Hier scheidt men de feitelijk gekozen techniek om in het geheugen verschillende waarden van een (abstracte) variabele te representeren van de voor de bewijsvoering meest conveniente wijze om deze verschillende waarden te karakteriseren, daarmee het correctheidsbewijs in dit opzicht factoriserend. Aan de hand van enige simpele voorbeelden zal een en ander geïllustreerd worden.

RECURSIEVE FUNCTIES EN TURING-MACHINES

door

A. Heyting

1. Primitief recursieve functies

De meest voor de hand liggende methode om een getaltheoretische functie te definiëren is die van de recursie. Men geeft $f(0)$ en een voorschrift om $f(x+1)$ te berekenen uit $f(x)$. Nauwkeuriger wordt deze methode als volgt vastgelegd.

Als uitgangsfuncties, die bekend worden ondersteld, kiezen wij

1. De opvolgerfunctie $f(x) = x+1$.
2. De constante functies $f(x_1,\dots,x_n) = a$, a een gegeven getal.
3. De projectiefuncties $f(x_1,\dots,x_n) = x_i$, i een der getallen $1,\dots,n$.

Nieuwe functies worden afgeleid door substitutie en door recursie.

Schema voor substitutie:

$$f(x_1,\dots,x_n) = g(h_1(x_1,\dots,x_n),\dots,h_m(x_1,\dots,x_n)),$$

waarin $g,h_1,\dots,h_m$ eerder gedefinieerde functies van de aangegeven aantallen veranderlijken zijn.

Schema voor recursie:

$$f(0,x_2,\dots,x_n) = g(x_2,\dots,x_n)$$

$$f(x_1+1,x_2,\dots,x_n) = h(f(x_1,\dots,x_n),x_1,\dots,x_n).$$

Een functie, die op deze wijze gedefinieerd kan worden, wordt tegenwoordig primitief recursief genoemd. De definitie van een dergelijke functie geeft meteen een methode aan, om de functiewaarde voor gegeven argumentwaarden uit te rekenen. Een functie, waarvoor een effectieve methode van berekening bestaat, wordt berekenbaar genoemd. Iedere primitief recursieve functie is dus berekenbaar.

Het is gemakkelijk voor de eenvoudigste functies uit de rekenkunde aan te tonen, dat zij primitief recursief zijn, maar ook ingewikkelder functies zoals de volgende behoren tot deze categorie:

$p(x)$ = het x^e getal in de naar grootte geordende rij der priemgetallen.

$\exp(x,y)$ = de exponent van het getal $p(x)$ in de ontbinding in priemfactoren van y.

Onder de representerende functie van een n-aire relatie R verstaat men de functie f_R van n veranderlijken met de volgende eigenschappen:

$$f(x_1,\dots,x_n) = 0 \quad \text{als } (x_1,\dots,x_n) \in R,$$

$$f(x_1,\dots,x_n) = 1 \quad \text{anders.}$$

R heet primitief recursief, als f_R primitief recursief is.

R heet beslisbaar, als er een effectieve methode is om voor een willekeurig gegeven n-tal $(x_1,\dots,x_n)$ uit te maken of het tot R behoort, dus als f_R berekenbaar is. Het is duidelijk, dat iedere primitief recursieve relatie beslisbaar is.

Alle functies, die men praktisch tegenkomst en die berekenbaar zijn, blijken primitief recursief te zijn. Ook verschillende veralgemeningen van het recursieschema voerden niet tot nieuwe functies. Toch slaagde Ackermann er in 1928 in een functie te definiëren, die berekenbaar is, maar niet primitief recursief. Het volgende voorbeeld van zo'n functie wordt verkregen door een methode, die ook in andere gevallen belangrijke resultaten geeft.

Laat V een verzameling recursieve functies zijn. De (n+1)-aire functie u heet een <u>universele functie</u> voor de n-aire functies uit V, als er voor elke n-aire functie f uit V minstens een getal a bestaat, zodat $f(x_1,\ldots,x_n) \equiv u(a,x_1,\ldots,x_n)$.

<u>Stelling</u>. Wanneer V de volgende voorwaarden 1), 2), 3) vervult, dan bevat V geen universele functie.

1) V bevat de opvolgerfunctie.
2) V bevat alle projectiefuncties.
3) V is gesloten tegenover substitutie.

<u>Bewijs</u>. Stel V bevat een universele functie u voor de functies van een veranderlijke uit V. Dan behoort volgens 2) en 3) ook de functie u(x,x) tot V, en volgens 1) en 3) ook u(x,x) + 1. Daar u een universele functie is, is er een getal a, zodat $u(x,x) + 1 \equiv u(a,x)$. Substitutie van a voor x geeft een contradictie.

<u>Gevolg</u>. Er is geen primitief recursieve universele functie voor de primitief recursieve functies van een veranderlijke.

Nu laat ik zien, dat er wel een berekenbare universele functie is voor de primitief recursieve functies van een veranderlijke. Daartoe merk ik op, dat elke primitief recursieve functie gedefinieerd wordt door een eindig aantal vergelijkingen. Wij kunnen dergelijke stellen vergelijkingen schrijven met behulp van een eindig alfabet, bijvoorbeeld:

f | / S 0 ( ) x = , .

Veranderlijken worden geschreven als x gevolgd door een aantal |, getallen als 0 gevolgd door een aantal |, functieveranderlijken als f gevolgd door een aantal | en een aantal / (dit laatste aantal geeft het aantal argumenten aan). S betekent de opvolgerfunctie. Het is gemakkelijk regels aan te geven om voor een gegeven rij tekens uit te maken of zij de volledige definitie van een primitief recursieve functie is.

Daar de verzameling van alle eindige rijen tekens uit het alfabet aftelbaar is, verkrijgt men zo een aftelling van alle definities van primitief recursieve functies. Laat g_n de functie zijn, die door de n^e definitie verkregen wordt. Dan is $g_n(x)$, beschouwd als functie van de twee veranderlijken n, x, een universele functie voor de primitief recursieve functies van een veranderlijke, en volgens haar ontstaan is zij berekenbaar.

2. Recursieve functies

Er was dus aanleiding te zoeken naar een meer algemene definitie van berekenbare functies. Dit is op verscheidene manieren met succes gedaan. Ik zal enige van die methoden kort schetsen en daarna op een enkele uitvoeriger ingaan.

1. Men kan afzien van de bijzondere vorm, die het stelsel vergelijkingen in het geval van de primitief recursieve functies heeft. Men noemt een functie recursief, als haar waarde voor gegeven argumentwaarden berekend kan worden uit een zeker stelsel vergelijkingen door toepassing van de volgende regels:

Substitutie van getallen voor veranderlijken.

Vervanging van $h(a_1,...,a_n)$ door het getal a, mits de vergelijking $h(a_1,...,a_n) = a$ tevoren is afgeleid.

(De lezer wordt verzocht hier en in het vervolg zich voor te stellen, dat alle hier verkort weergegeven vergelijkingen met behulp van de tekens van het gekozen alfabet geschreven zijn.)

2. Men voegt aan de regels voor de primitief recursieve functies de regel voor de minimalisatie toe. Deze luidt als volgt:

Laat f een binaire functie zijn, zodanig, dat er voor ieder getal a minstens een getal b bestaat, zodat $f(a,b) = 0$. Het kleinste getal met deze eigenschap stellen wij voor door $\mu y(f(a,y) = 0)$. De functie $\mu y(f(x,y) = 0)$ wordt aan het stelsel verkregen functies toegevoegd. Iedere zo verkregen functie heet μ-recursief.

3. Men stelt een algemene definitie op van een algoritme, d.w.z. een methode, waardoor uit gegeven tekenrijen nieuwe tekenrijen afgeleid kunnen

worden. Een algoritme, die van n getallen als uitgangsrij tot een getal voert, definieert een n-aire functie. Alle zo verkregen functies heten calculabel.

4. Men kan een machine definiëren die, wanneer n getallen als input gegeven worden, een getal als output voortbrengt. Een klasse van dergelijke machines is door Turing gedefinieerd. Een functie, die door zo'n machine berekend wordt, heet Turing-recursief.

Het is gebleken, dat de vier geschetste methoden tot hetzelfde resultaat voeren: Iedere functie, die volgens een der methoden berekenbaar is, is dit ook volgens alle andere. De functie heet dan recursief.

3. Turing-machine

Wij zullen de methode der Turing-machines wat uitvoeriger bespreken. Een Turing-machine bevat een in principe naar weerszijden oneindig lange band, verdeeld in vakjes. In ieder vakje kan de machine een der tekens van het alfabet $S_0, S_1, \ldots, S_k$ schrijven. Er zijn l mogelijke inwendige toestanden $q_1, \ldots, q_l$ van de machine. De machine werkt in achtereenvolgende stappen. Op ieder ogenblik is zij gericht op een bepaald vakje van de band, dat ik het schrijfvakje zal noemen. Wat de machine bij een bepaalde stap gaat doen, hangt af van de inwendige toestand en van het teken dat in het schrijfvakje staat. Een stap bestaat uit drie delen:

1. Het teken in het schrijfvakje wordt door een ander (of hetzelfde) teken vervangen.
2. De machine richt zich op een vakje, dat het vakje links van het schrijfvakje, het schrijfvakje zelf of het vakje rechts van het schrijfvakje kan zijn; dit vakje wordt het schrijfvakje voor de volgende stap.
3. De machine gaat in een nieuwe inwendige toestand over.

De machine wordt dus volledig beschreven door een eindig aantal regels van de vorm

$$q_i S_j \to S_h T q_m,$$

waarin T een der letters L(inks), R(echts) of C(onstant) is. Verkeert de machine in toestand q_i en staat in het schrijfvakje S_j, dan verandert zij S_j in S_h, voert de verschuiving T uit en gaat over in toestand q_m. Komt de combinatie $q_i S_j$ in geen enkele regel links voor, dan stopt de machine. Dezelfde combinatie mag natuurlijk niet in twee regels links voorkomen.

In het vervolg is S_0 steeds het lege vakje, S_1 het teken 1. Het getal n wordt voorgesteld door n+1 achtereenvolgende tekens 1. Laat nu $\mathcal{T}$ een Turingmachine zijn. Wij kunnen $\mathcal{T}$ als volgt een functie f laten berekenen.

Het getal x op de band wordt voorgesteld door een rij van x+1 achtereenvolgende vakjes met het symbool 1. Een rij getallen wordt voorgesteld door deze blokken enen achter elkaar te schrijven, van elkaar gescheiden door blanco vakjes, terwijl de rest van de band leeg is.

Om f(x) te vinden richten we de machine in toestand q_1 op het meest linkse vakje, waarin een 1 behorende bij het argument x staat. Als de machine gebruikt wordt om een functie in meerdere veranderlijken te berekenen dan richten we de machine op de eerste 1 van het eerste argument. Nadat de machine gestopt is bekijken we het laatste blok tekens 1 op de band. Dit blok stelt de uitkomst f(x) (of $f(x_1,\dots,x_n)$) voor.

Het is duidelijk, dat men geen machine nodig heeft om deze bewerkingen uit te voeren. Men kan het net zo goed op het papier doen. De spreekwijze der Turing-machines versterkt de suggestie dat wij met een machinaal verlopend proces te maken hebben. De vraag naar de praktische uitvoering speelt geen rol.

Voorbeelden van Turing-machines

De regels voor elke machine zijn gegeven in de vorm van een tabel.

Machine $\mathcal{A}$ telt bij het laatste getal op de band 1 op.

	0	1
q_1	$1Lq_2$	$1Rq_1$
q_2	$0Rq_0$	$1Lq_2$

Machine *F* trekt van het laatste getal op de band 1 af.

	0	1
q_1	$0Lq_2$	$1Rq_1$
q_2		$0Lq_3$
q_3	$0Rq_0$	$1Lq_3$

Machine *D* verschuift, als een rij getallen op de band staat, het schrijfvakje een getal naar links.

	0	1
q_1	$0Lq_2$	$1Lq_1$
q_2	$0Lq_2$	$1Lq_3$
q_3	$0Rq_0$	$1Lq_3$

```
1 1 1 1 0 0 0 1 1 1    wordt
              q1

1 1 1 1 0 0 0
q0
```

(Onder het teken in het schrijfvakje is de inwendige toestand gedrukt.)

Machine *G* verschuift evenzo een getal naar rechts.

	0	1
q_1	$0Rq_2$	$1Rq_1$
q_2	$0Rq_2$	$1Cq_0$

```
1 1 1 1 0 0 0 1 1 1    wordt
q1

1 1 1 1 0 0 0 1 1 1
              q0
```

Machine *B*, gericht op het laatste getal op de band, schrijft daar achter het getal 0.

	0	1
q_1	$0Rq_2$	$1Rq_1$
q_2	$1Cq_0$	

```
1 1 1 1 0 0 0    wordt
q1

1 1 1 1 0 1 0
          q0
```

Machine *H*, gericht op een getal, vult dat getal aan met tekens 1, tot het nog slechts één teken 0 van het volgende getal gescheiden is.

	0	1
q_1	$1Rq_2$	$1Rq_1$
q_2	$1Rq_2$	$1Lq_3$
q_3		$0Lq_4$
q_4	$0Rq_0$	$1Lq_4$

$\underset{q_1}{1}$ 1 1 0 0 0 1 1 1 wordt

$\underset{q_0}{1}$ 1 1 1 1 0 1 1 1

Het is eenvoudig, bij twee machines T_1 en T_2 een machine $U = T_1T_2$ te construeren, die hetzelfde resultaat geeft als wanneer de output van T_1 als input voor T_2 gebruikt wordt. Als voorbeeld schrijf ik het schema $B\ A$ op:

	0	1
q_1	$0Rq_2$	$1Rq_1$
q_2	$1Cq_3$	
q_3	$1Lq_4$	$1Rq_3$
q_4	$0Rq_0$	$1Lq_4$

$\underset{q_1}{1}$ 1 1 1 0 0 0 0 wordt

1 1 1 1 0 $\underset{q_0}{1}$ 1 0

In het schema van B is q_0 vervangen door q_3; in dat van A is q_1 vervangen door q_3 en q_2 door q_4. De algemene regel voor de vorming van T_1T_2 is aan de hand van dit voorbeeld gemakkelijk te formuleren, maar enigszins omslachtig.

Men kan nu ook langere reeksen van machines koppelen; hierbij is ook terugkoppeling mogelijk. Om de machine $T\ \underleftarrow{U\ V\ W}$ te definiëren, maken wij eerst het schema voor $T\ U\ V\ W$ en vervangen daarna de eindtoestand q_0 van W door de begintoestand van U. Het is duidelijk, dat deze machine eeuwig door zal lopen, tenzij een inrichting ingebouwd wordt, die hem op zeker ogenblik doet stoppen. Dit kan door middel van de volgende machine E.

	0	1
q_1		$1Rq_2$
q_3	$0Lq_3$	$1Lq_4$

0 0 $\underset{q_1}{1}$ 0 0 wordt 0 0 $\underset{q_3}{1}$ 0 0

0 0 $\underset{q_1}{1}$ 1 . . . wordt 0 0 $\underset{q_4}{1}$ 1 . . .

E gaat in toestand q_3 over als ze op het getal 0 gericht is, maar in q_4 als ze op een ander getal gericht is. Men kan nu aan q_3 machine T en aan

q_4 machine U koppelen. Dit geef ik aan door het schema $E \left| \begin{matrix} T \\ U \end{matrix} \right.$

Voorbeeld: $E \left| \begin{matrix} A \\ F \end{matrix} \right.$

	0	1
q_1		$1Rq_2$
q_2	$0Lq_3$	$1Lq_4$
q_3	$1Lq_5$	$1Rq_3$
q_5	$0Rq_0$	$1Lq_5$
q_4	$0Lq_6$	$1Rq_4$
q_6		$0Lq_7$
q_7	$0Rq_0$	$1Lq_7$

Deze machine telt 1 op bij 0 en trekt 1 af van ieder ander getal:

$\underset{q_1}{1}\ 0\ 0\ 0\ 0$ wordt $\underset{q_0}{1}\ 1\ 0\ 0\ 0$

$\underset{q_1}{1}\ 1\ 1\ 0\ 0$ wordt $1\ 1\ 0\ 0\ 0$

De tot hiertoe geconstrueerde machines zijn voldoende om te bewijzen, dat iedere μ-recursieve functie Turing-recursief is. Daartoe moeten machines aangegeven worden, die de uitgangsfuncties berekenen, en verder moet voor ieder schema aangegeven worden, hoe uit de machines voor de gegeven functies een machine voor de gedefinieerde functie verkregen wordt.

Ik construeer eerst voor ieder natuurlijk getal $m > 0$ een machine C_m, die, als een rij van minstens m getallen op de band staat, terwijl ze gericht is op het laatste daarvan, achter die rij het m^{de} getal uit de rij (van achteren af gerekend) copieert.

$$C_m = B\,D^m\,E \left| \begin{matrix} H\,G^m \\ F\,G^m\,A. \end{matrix} \right.$$

(met een pijl van A terug naar D^m)

Merk op dat de hiervoor geconstrueerde machines de eigenschap hebben dat ze het bandgedeelte links van het eerst gelezen getal na afloop in de oorspronkelijke toestand achterlaten.

De machines voor de uitgangsfuncties zijn gemakkelijk aan te geven. Laat het schema voor een recursie met twee veranderlijken zijn:

$$f(0,y) = g(y),$$

$$f(x+1,y) = h(x,y,f(x,y)).$$

De machines M_g en M_h, die g en h berekenen, zijn gegeven. M_g en M_h zijn zo geconstrueerd dat ze de argumenten in de oorspronkelijke toestand achter laten en na afloop op het resultaat gericht zijn dat direct achter de argumenten staat. Een machine M_f, die f berekent, wordt als volgt verkregen:

```
M_f = M_g C_3 E | C_2
                | B C_4^2 D^2 M_h C_5 F E | C_2
                          ↑               | C_5 A C_5 C_4.
                          └──────────────────────────┘
```

$\mathcal{D}^2$ betekent: tweemaal $\mathcal{D}$ achter elkaar.
Op de band worden de getallen x en y geschreven, gescheiden door een teken 0. De machine is gericht op y. Voor x = 2 schrijft de machine

2 y g(y) 2 0 y g(y) f(1,y) 1 1 y f(1,y) f(2,y) 0 f(2,y).

Nauwkeuriger gezegd: ze schrijft de door deze termen voorgestelde getallen op.

Laat f gedefinieerd zijn door $f(x) = \mu y(g(x,y) = 0)$, dan is

```
M_f = B D M_g E | C_2
          ↑     | C_3^2 A D.
          └─────────────┘
```

Het wordt aan de lezer overgelaten, de schema's voor de substitutieregel en voor de definitie van functies van meer veranderlijken op te schrijven.

4. <u>These van Church</u>

Het feit, dat zeer verschillende definities tot hetzelfde begrip leiden, bewijst dat dit begrip belangrijk is. Het is een van de argumenten voor de bewering die als <u>these van Church</u> bekend staat, namelijk: <u>Iedere berekenbare functie is recursief</u>. Het is niet in te zien hoe men een dergelijke bewering zou kunnen bewijzen, maar er is evenmin enige aanwijzing in welke richting men naar een berekenbare, niet recursieve functie zou moeten zoeken.

De vraag of iedere recursieve functie berekenbaar is, wordt zelden

gesteld; het schijnt vanzelf te spreken. Toch is er aanleiding, er even op in te gaan. Een eenvoudig voorbeeld toont, dat het twijfelachtig is of iedere recursieve relatie beslisbaar is. Denk aan de volgende redenering: Iedere eindige verzameling is (primitief) recursief. Iedere deelverzameling van een eindige verzameling is eindig, dus recursief. Dus als iedere recursieve verzameling beslisbaar is, dan is iedere deelverzameling van een eindige verzameling beslisbaar. Dit is zeker niet waar. De oplossing van deze paradox is als volgt. Men kan eerst beweren dat een recursieve functie berekenbaar is, als men de Turing-machine voor die functie werkelijk construeren kan, niet, als men alleen abstract bewezen heeft, dat die machine moet bestaan. Dit houdt in dat voor een eindige verzameling precies bekend moet zijn welke elementen er toe behoren, met andere woorden, men moet al weten dat de verzameling beslisbaar is. Samenvattend: Een recursieve functie is alleen berekenbaar, als haar Turing-machine effectief aangegeven kan worden.

5. Partieel recursieve functies

Laat men de machine voor $f(x) = \mu y(g(x,y) = 0)$ werken op een getal x waarvoor geen y bestaat, zodat $g(x,y) = 0$, dan loopt de machine eeuwig door. $f(x)$ is niet gedefinieerd. In het algemeen kan men zeggen, dat een willekeurige Turing-machine een functie definieert, die slechts bepaald is op een deelverzameling van de natuurlijke getallen, die ook leeg kan zijn. Een dergelijke functie heet partieel recursief. Ook de andere definities van recursieve functies kunnen tot partiële functies uitgebreid worden; zij leiden tot equivalente begrippen.

Bij een gegeven alfabet $S_0,\ldots,S_k$ en gegeven inwendige toestanden $q_1,\ldots,q_l$ is slechts een eindig aantal Turing-machines mogelijk. Bijgevolg kan men alle mogelijke Turing-machines nummeren. Wij denken een vaste nummering gegeven en stellen de functie die door de n^e machine T_n gedefinieerd wordt voor door F_n. Natuurlijk komt elke partieel recursieve functie in deze rij oneindig vaak voor. Het definitiegebied van een partieel recursieve functie heet een recursief opsombare verzameling. Het definitiegebied van F_n stellen wij voor door W_n.

Een berekening door een Turing-machine bestaat uit een aantal woorden op de band, ieder met een schrijfvakje en een inwendige toestand, kort gezegd, uit een aantal machinewoorden. Het is eenvoudig, alle eindige rijen machinewoorden bij een gegeven machine te nummeren; hieronder komen alle berekeningen voor, maar ook andere rijen. Nu voer ik de relatie $T(n,x,y)$ in met de betekenis: y is het nummer van een berekening van $\mathcal{T}_n$ met als input x. Men kan bewijzen, dat deze relatie bij geschikt gekozen aftelling primitief recursief is in de veranderlijken n, x, y.

6. Recursief onoplosbare problemen

Beschouw de verzameling $K = \{x \mid x \in w_x\}$. Neem aan dat deze verzameling recursief is, en noem haar representerende functie f. Definieer g door: $g(x) = 1$ als $f(x) = 1$; $g(x)$ is onbepaald als $f(x) = 0$. Het is gemakkelijk uit een Turing-machine voor f er een voor g te maken, dus g is partieel recursief. Stel $g = F_n$. Nu heeft men enerzijds, wegens de definitie van w_n,

$$F_n(n) \text{ bepaald} \longleftrightarrow n \in w_n,$$

anderzijds, wegens de definitie van g,

$$F_n(n) \text{ bepaald} \longleftrightarrow g(n) \text{ bepaald} \longleftrightarrow f(n) = 1 \longleftrightarrow n \notin w_n.$$

Hieruit zou volgen, dat $F_n(n)$ voor geen waarde van n bepaald kan zijn, maar dit is in strijd met het bestaan van overal bepaalde recursieve functies. De conclusie is, dat K niet recursief is.

Wij hebben hier een eerste voorbeeld van een niet recursief oplosbaar probleem. Laat P een predicaat van natuurlijke getallen zijn. Men zegt, dat het probleem of een getal x de eigenschap P heeft, recursief oplosbaar is, als P recursief is.

Ook de relatie $x \in w_y$ is niet recursief. Sterker: er is een getal m zodat $x \in w_m$ niet recursief is. Voor de Turing-machine $\mathcal{T}_m$ is het "halting problem" recursief onoplosbaar. Voor $\mathcal{T}_m$ kan men de machine nemen, die $F_m = \mu y T(x,x,y)$ berekent, want $F_m(x)$ is bepaald als $x \in w_x$.

Voor verschillende wiskundige problemen is bewezen, dat zij recursief onoplosbaar zijn. Deze resultaten behoren tot de meest spectaculaire van het grondslagenonderzoek. De meeste bewijzen worden geleverd door reductie tot het hierboven genoemde resultaat.

In 1900 hield Hilbert op het internationale wiskundecongres in Parijs een voordracht getiteld "Mathematische Probleme". Zijn 10^{e} probleem was het volgende. Gegeven een veelterm in n veranderlijken $P(x_1,\ldots,x_n)$ met gehele coëfficiënten; te onderzoeken of de vergelijking $P(x_1,\ldots,x_n) = 0$ een oplossing in gehele getallen heeft. Men heeft lange tijd gezocht naar een bewijs voor de recursieve onoplosbaarheid van dit probleem. Na veel voorbereidend werk van verscheidene wiskundigen deed Matiasewitsj in 1970 de laatste stap.

Een ander voorbeeld is het woordprobleem voor groepen. Laat een groep gegeven zijn door een eindig aantal voortbrengenden en een eindig aantal relaties; te onderzoeken of een gegeven product van voortbrengenden gelijk is aan het eenheidselement van de groep. De recursieve onoplosbaarheid van dit probleem werd in 1955 bewezen door Novikov en onafhankelijk van hem in 1957 door Boone. Om dit probleem onder de definitie van recursiviteit te brengen, zou men eerst de groepentheorie moeten arithmetiseren. Bij gebruik van Turing-machines is dit niet nodig. Men kan de vraag zo formuleren: Is er een Turing-machine, die als alfabet het stelsel voortbrengenden van de groep heeft en, gevoed met een willekeurig woord, 0 als uitkomst geeft wanneer dat woord het eenheidselement is en 1 anders. Ook hier geldt de stelling weer in de strenge zin: Er is een groep, waarvoor het woordprobleem recursief onoplosbaar is.

In de logica is het decisieprobleem van een axiomatische theorie het belangrijkste voorbeeld. Een axiomatische theorie is gegeven door een alfabet en een eindig aantal woorden die als axioma's gelden. Verder zijn er de formele deductieregels van de logica, waarmee stellingen afgeleid worden. De theorie wordt gearithmetiseerd door alle formules en rijen formules te nummeren. De vraag of een bepaalde formule een stelling is, gaat dan over in een rekenkundig probleem. Verschillende theorieën zijn nier recursief beslisbaar. Men behoeft zich deze vraag niet te beperken tot eindig geaxiomatiseerde theorieën. Het belangrijkste resultaat is, dat de reken-

de rekenkunde, gebaseerd op de axioma's van Peano, niet recursief beslisbaar is.

Uit het bestaan van recursief onbeslisbare axiomatische theorieën volgt dat ook het decisieprobleem van de elementaire logica niet recursief beslisbaar is. Dit is voor het eerst door Church bewezen in 1936.

7. Hiërarchieën

Na de recursieve zijn de recursief opsombare verzamelingen de eenvoudigste. Zij worden gegeven door een voorwaarde van de vorm $\bigvee y R(x,y)$, waarin R een recursieve relatie is. Men kan nu een hiërarchie van relaties opbouwen, door steeds meer quantoren voor een recursieve relatie te plaatsen. Daarbij kan men zich beperken tot afwisselend een $\bigvee$- en een $\bigwedge$-teken. Een verzameling behoort tot Σ_n als de definitie n quantoren bevat, die met $\bigvee$ beginnen; analoog wordt Π_n gedefinieerd. $\Sigma_0 = \Pi_0$ = de verzameling der recursieve verzamelingen. Σ_1 is de verzameling der recursief opsombare verzamelingen.

Een andere hiërarchie wordt verkregen door relativering van het begrip recursieve functie met betrekking tot een verzameling A. Men stelt zich voor, dat men de Turing-machine T op ieder moment kan stoppen om na te gaan of het op dat ogenblik verkregen resultaat tot A behoort. Is dat het geval, dan gaat ze over in toestand q_i, anders in toestand q_j. De op deze wijze door T gedefinieerde functie heet partieel A-recursief. Stopt de machine voor het onderzoek wanneer zij in toestand q_k gericht is op het teken S_m, dan kan dit aangegeven worden door een regel $q_k S_m \rightarrow q_i q_j$. Het is nu weer mogelijk, alle zo gerelativeerde Turing-machines te nummeren en de relatie $T^A(n,x,y)$ te definiëren, die betekent: y is het nummer van een berekening van T_n^A met als input x. De door T_n^A gedefinieerde functie is voor het argument x bepaald als $\bigvee y T^A(n,x,y)$; de verzameling waarop dit geldt is w_n^A. Analoog met K definiëren wij:

$$A' = \{x \mid x \in w_x^A\}.$$

De overgang van A op A' heet "jump". Door herhaling van de jump krijgt men een hiërarchie van verzamelingen.

Is B, A-recursief en A, B-recursief, dan hebben A en B dezelfde onoplosbaarheidsgraad. De onoplosbaarheidsgraden vormen een distributieve tralie (lattice); het onderzoek naar de eigenschappen van die tralie is een belangrijk onderwerp in de recursietheorie.

8. Recursieve analyse

Laat een nummering van de rationale getallen gegeven zijn: $r_1, r_2, \dots$. Een reëel getal is gegeven door een rij $r_{a_1}, r_{a_2}, \dots$. Voor een recursief reëel getal stelt men de volgende eisen:

Er is een recursieve functie f zodat $a_n = f(n)$ voor iedere n.

Er is een recursieve functie g zodat $|r_{a_n} - r_{a_m}| < 2^{-k}$ voor $n,m > g(k)$.

Op deze basis is een groot stuk recursieve analyse opgebouwd, vooral door Russische wiskundigen.

Het is niet mogelijk, in deze voordracht op de laatstgenoemde onderwerpen uitvoerig in te gaan. Daarvoor moet naar de volgende literatuur verwezen worden.

Referenties

Davis, M., *Computability and unsolvability,* McGraw-Hill, New York, London, (1958).

Davis, M., *Hilbert's tenth problem is unsolvable,* Am. Math. Man., 80 (1973), 233-269.

Grzegorczyk, A., *Fonctions recursives,* Gauthier-Villars, Paris, (1961).

Kleene, S.C., *Introduction to metamathematics,* North-Holl. Publ. Co., Amsterdam, (1952).

Matiasewitsj, Joe. V., *Diofantonostj peretsjislimych mnozjestv,* Doklady Akad. Naoek USSR 191 (1970), 279-282;
Engelse vertaling: Soviet Math. Doklady 11, 354-357.

Rogers, H., *Theory of recursive functions and effective computability,* McGraw-Hill, New York, London, (1967).

Shoenfield, J.R., *Degrees of unsolvability,* North-Holl. Publ. Co., Amsterdam, (1971).

KRISTALSTRUKTUREN, EEN EXPERIMENT IN COMPUTER-KUNST

door

L.J.M. GEURTS

Inleiding

In de jaren 1968 tot 1973 zijn op het Mathematisch Centrum door de auteur van dit artikel en L.Meertens een aantal experimenten gedaan op het gebied van computer-kunst. Deze projekten bestreken de terreinen van geluid, beeld en tekst.

In 1968 behaalde Meertens met zijn strijkkwartet in klassieke stijl een eervolle vermelding in de Computer-Composed Music Competition, georganiseerd door IFIP (International Federation for Information Processing). Een uiteenzetting over het strijkkwartet en een grammofoonplaatje met de muziek, gespeeld door het Amsterdams Strijkkwartet zijn te vinden in [5]. De partituur is afgedrukt in [6].
Samen met de componisten Louis Andriessen en Peter Schat leverden Geurts en Meertens in 1969 een bijdrage aan de opera Reconstructie, en met Andriessen realiseerden zij de Sonate Opus 2 nr.1, gebaseerd op Beethoven's gelijknamige compositie.
De drie genoemde muzikale projekten zijn beschreven in [2].

In 1970 schreven Geurts en Meertens een programma dat de bedoeling had "boeiende" strukturen voort te brengen, die later zowel om de manier, waarop ze gevormd werden, als om hun visuele eigenschappen "kristalstrukturen" werden genoemd. Deze kristalstrukturen werden op een aantal plaatsen in Nederland geëxposeerd en behaalden prijzen in de 8th Annual Computer Art Contest van het tijdschrift Computers and Automation [1], en in het Third Annual ACM Computer Arts Festival. Het kristalstrukturen-projekt wordt in dit

artikel uitvoerig besproken. Een eerdere uiteenzetting is verschenen in [3], [4] en [8].

Van de andere experimenten kunnen genoemd worden:

- het "modulenplot"-programma, dat de toeschouwer-gebruiker de gelegenheid biedt uit een aantal eenvoudige vierkante motieven er enkele te kiezen, waarna het programma een tableau van de gekozen vierkanten tekent.
- het "orakel"-programma, dat grammaticaal correcte teksten produceert, die bij oppervlakkige beschouwing de indruk van mystiek en erudiet georakel geeft.
- een programma, dat een klasse van gestileerde letterachtige vormen genereert [7].
- het Volkslied (met Andriessen), een projekt, waarin een melodie stapsgewijs in een andere melodie wordt getransformeerd.

Procedurele kunst

De term "computerkunst" geeft slechts een gebrekkige aanduiding van de aard en de bedoeling van de hierboven genoemde experimenten. Het gebruik van de computer is niet werkelijk essentieel; sommige van de projekten zouden zelfs gemakkelijk zonder computer gerealiseerd kunnen zijn, voor andere zou wat meer tijd nodig geweest zijn om ze "met de hand" te verwezenlijken. Wel essentieel is dat in alle gevallen het resultaat nadrukkelijk volgens een bepaalde procedure wordt voortgebracht. Er wordt niet eerst een bepaald resultaat ontworpen en vervolgens een methode bedacht, die dit resultaat gegarandeerd zal opleveren. De procedure is wat in feite door de "computerkunstenaar" gecreëerd wordt, niet de afzonderlijke resultaten die weer door die procedure gegenereerd worden. Het is dan ook misschien beter te spreken van "procedurele kunst".
Het ligt voor de hand dat bij het beoordelen van de waarde van zo'n procedure andere criteria gehanteerd worden dan voor kunst in het algemeen: criteria als eenvoud, algemene toepasbaarheid en elegantie.

Het woord "kunst" in de term "computerkunst" (en in "procedurele kunst") kan ook gemakkelijk misverstand wekken. Door de beperking tot resultaten die verkregen kunnen worden door een nauwkeurig, vooraf geconstrueerd proces in werking te zetten, is verreweg het grootste gebied van de

bestaande kunst onbereikbaar. Het is voorlopig ondenkbaar dat een schilderij uit het Rijksmuseum, of een bestaande operette, western of stripverhaal langs automatische weg ontworpen zou kunnen worden. Vooralsnog moet procedurele kunst beperkt blijven tot kunstvormen, waarin de vorm veel belangrijker is dan de betekenis: abstracte beeldende kunst, muziek, ballet en sterk gereduceerde vormen van poëzie.
Op enkele van deze gebieden sluit de procedurele computerkunst goed aan bij bepaalde bestaande werkwijzen: in de muziek de aleatoriek en het twaalftoonsysteem, in de beeldende kunst het gebruik van het toeval en het methodisch constructivisme.

Toeval

Zoals in zeer veel voorbeelden van computerkunst wordt in de genoemde projekten een grote rol toebedeeld aan het toeval. Een eerste reden hiervoor is, dat het algemene idee dat aan een bepaald projekt ten gronslag ligt dikwijls nog niet elk detail van het resultaat vastlegt. Dan kan dat idee weliswaar zodanig aangevuld worden dat het resultaat wel volledig gedefinieerd is, maar er is dan vaak een scala van mogelijkheden, waarbinnen het kiezen moeilijk, en ook irrelevant, is. Elk van deze keuzen zou een verschillend resultaat geven, maar het oorspronkelijke idee zou steeds worden teruggevonden: de stijl zou gelijk blijven. In zo'n situatie kan heel goed het doen van een toevallige keuze (of een serie toevallige keuzen) in de procedure ingebouwd worden. Een bijkomend voordeel daarvan is, dat hetzelfde proces dan in staat is steeds weer nieuwe (hoewel in stijl gelijke) resultaten voort te brengen.

Een andere reden om het toeval in de te volgen procedure te laten meespelen is gelegen in het volgende. Een van de oogmerken bij elk van de experimenten is geweest het tot stand brengen van een boeiend resultaat, in die zin dat het in staat zou zijn de aandacht van de toeschouwer (of luisteraar of lezer) enige tijd vast te houden. Een eenvoudige, streng doorgevoerde regelmaat is hiervoor niet geschikt: zodra de toeschouwer de regelmaat doorzien heeft is er voor hem niets nieuws meer te ontdekken. Twee andere mogelijkheden dienen zich aan: een ingewikkelde regelmaat kiezen, of juist een onregelmatige opbouw.

Een ingewikkelde regelmaat heeft het nadeel dat deze in het algemeen

niet als regelmaat gepercipieerd zal worden, zodat de moeite, die eraan besteed is om die ingewikkelde regelmaat op te bouwen, vergeefs is geweest. Om tot een boeiend resultaat te komen lijkt een onregelmatige opbouw dus wenselijk, en zo'n opbouw kan met behulp van het toeval op allerlei manieren gerealiseerd worden.

Bij de keuze voor een onregelmatige struktuur moet wel de grauwheid van volstrekte chaos vermeden worden: als een toeschouwer de helft van een geheel regelmatig beeld te zien krijgt, kent hij daarmee de andere helft ook; krijgt hij de helft van een volstrekt onregelmatig beeld te zien, dan is hij in de andere helft al evenmin geïnteresseerd, omdat die er wel even rommelig zal uitzien. Twee geheel chaotische beelden zijn eigenlijk even moeilijk van elkaar te onderscheiden als twee geheel identieke beelden.

De bezwaren, die aan geheel regelmatige en geheel onregelmatige patronen kleven, kunnen worden weggenomen door beide alternatieven te combineren. Chaos en regelmaat kunnen elkaar goed aanvullen: regelmaat trekt de aandacht en lokt een hypothese over de opbouw uit; chaos ontkent die hypothese en geeft een speels karakter aan het totaal. Voorbeelden van boeiende strukturen van dit type komen in de natuur veelvuldig voor: het patroon van golven op het water, de struktuur van bomen, de geluiden van vogels, vingerafdrukken, het craquelé van opgedroogde modder, enz.

Geprogrammeerd toeval

Waar in het voorgaande sprake was van het toepassen van het toeval in een proces, werd daarmee niet het soort menselijke willekeur bedoeld dat bij het aanwijzen van een willekeurig woord in een boek een bladzijde kiest ergens halverwege het boek en vervolgens ergens midden in een regel halverwege die bladzijde het vereiste toevallige woord aanwijst. Wel bedoeld wordt het strikte soort willekeur van een opgegooide munt of een geworpen dobbelsteen. In die opvatting kan men in een rij van 57 woorden een willekeurig woord aanwijzen door een telefoonboek op een of andere bladzijde (b.v. halverwege) te openen en van de telefoonnummers op die bladzijde het eerste te kiezen, waarvan de laatste twee cijfers een getal tussen 1 en 57 vormen. Dat getal verwijst dan naar een willekeurig woord in de strikte opvatting van het toeval: elk van de 57 woorden heeft een gelijke kans gekozen te worden (we

mogen aannemen dat elke combinatie van twee eindcijfers even vaak voorkomt en dat de telefoonnummers bij alfabetisch opeenvolgende namen geen speciaal verband hebben).

Het bepalen van een willekeurige volgorde van een aantal elementen kan zo gebeuren: schrijf elk van de elementen op een kaartje, doe de kaartjes in een doos, schud de doos goed en neem de kaartjes blindelings één voor één uit de doos. De volgorde, waarin de kaartjes uit de doos genomen worden, bepaalt dan een willekeurige volgorde van de elementen.

Kristalstrukturen

De bedoeling van dit projekt was een eenvoudige methode te geven om patronen te ontwerpen die regelmaat en onregelmatigheid op een natuurlijke manier in zich verenigen. De methode moest er niet alleen voor zorgen dat het resulterende patroon deels regelmatig en deels onregelmatig zou zijn, maar moest ook een organische samenhang tussen beide componenten garanderen. Zo'n samenhang wordt niet bereikt door uit te gaan van een geheel regelmatig patroon, en hierin vervolgens willekeurige verstoringen aan te brengen. In dat geval zou het resultaat wel half-regelmatig zijn, maar zou de gewenste samenhang ontbreken. Die aanpak is te vergelijken met het bekijken van een regelmatig beeld door een vuile ruit: er is geen verband tussen beide componenten van het uitzicht.

Het algemene idee, dat voor dit projekt is gekozen, is nu: ga uit van een onregelmatig patroon en versterk geleidelijk de regelmatige trekken die daarin van nature voorkomen.

Dit idee is natuurlijk nog zeer vaag; er zullen nog besluiten moeten worden genomen over de aard van het onregelmatige beginpatroon, over de soorten regelmaat, over de manier van versterken van de regelmaat, over het moment van ophouden met versterken.

a. Als beginpatroon werd een vierkant tableau van zwarte en witte vierkantjes gekozen, waarbij voor elk vierkantje met gelijke kansen zwart of wit geloot werd (vgl. het eerste tableau van fig. 8). Gegeven het uitdrukkingsmiddel van zwarte en witte vierkantjes is dit de manier om een volstrekt wanordelijke situatie te verkrijgen.

b. Hoe groter zo'n onregelmatig tableau is, des te meer soorten regelmaat zijn erin aan te wijzen: witte gebiedjes en zwarte gebiedjes, horizontaal gestreepte gebiedjes, stukjes schaakbord, enz. Een uniforme manier om dergelijke regelmaten te beschrijven is het geven van het verband tussen de kleur van een vierkantjes en de kleuren van vierkantjes in de omgeving. Enige voorbeelden van zulke beschrijvingen:

visuele regelmaat	beschrijving
witte en zwarte gebieden	een vierkantje heeft dezelfde kleur als zijn omgeving
horizontale strepen	een vierkantje heeft dezelfde kleur als zijn linker- en rechterburen, maar een andere dan zijn directe boven- en benedenbuur
schaakbord	een vierkantje heeft een andere kleur dan zijn vier directe buren

Het is bij deze regelmaten dus mogelijk een verband tussen de kleur van een vierkantje en de kleur van vierkantjes in zijn omgeving te geven, dat opgaat voor elk vierkantje, ongeacht zijn positie op het tableau.
Een andere eigenschap van deze regelmaten is, dat wit en zwart symmetrisch behandeld worden. Tot dit type regelmaten hebben we ons beperkt.

c. Er moet nu nog een methode gevonden worden, die in staat is de gekozen regelmaat te versterken in het chaotische beginpatroon, wélke regelmaat ook gekozen is. Die methode moet er vooral voor zorgen dat een enkel "stout" vierkantje in een gebied waar de gekozen regelmaat heerst, van kleur veranderd wordt. Als de regelmaat van een schaakbord gekozen is, zal een zwart vierkantje met vier zwarte buren wit gemaakt moeten worden; het ligt voor de hand een zwart vierkantje, dat drie zwarte en een witte buur heeft, ook wit te maken; heeft een zwart vierkantje twee zwarte en twee witte buren dan heeft veranderen weinig zin. Een eenvoudige methode om dit te bereiken kan zo beschreven worden: bij het bepalen van de nieuwe kleur van een vierkantje hebben zijn vier buren elk een stem: zwarte buren stemmen voor wit, witte buren stemmen voor zwart. (Dit stemgedrag wordt schematisch weergege-

ven in recept 2 (fig. 9). De min-tekens duiden een stem aan voor het tegengestelde van de eigen kleur.) De meerderheid beslist nu; als de stemmen staken blijft de toestand onveranderd.

De gekozen regelmaat kan dus versterkt worden door elk vierkantje van het tableau op deze wijze onder de loep te nemen. De volgorde, waarin de vierkantjes aan de beurt komen, is hierbij wel van invloed op het resultaat, omdat een vierkantje, zolang het nog niet aan de beurt geweest is, een andere invloed op zijn omgeving heeft dan wanneer het eenmaal van kleur is veranderd. Om stelselmatige beïnvloeding door deze volgorde uit te sluiten, is het proces zo ingericht, dat alle vierkantjes in een willekeurige volgorde eenmaal onder de loep genomen worden.

Laten we de gang van zaken zoals die hierboven beschreven is nu eens op de voet volgen aan de hand van een klein voorbeeld. In fig. 1 is een patroon van 7 × 7 vierkantjes getekend die willekeurig wit of zwart gemaakt zijn. Als regelmaat, die we hierin gaan versterken, kiezen we die van een schaakbord. Omdat daartoe van elk te veranderen vierkantje de kleur van zijn vier directe buren in beschouwing genomen moet worden, komen alleen de binenste 5 × 5 vierkantjes voor verandering in aanmerking. Deze 25 vierkantjes worden nu in willekeurige volgorde genummerd met de getallen 1 t/m 25 (fig. 2). Vierkantje 1 wordt nu onder de loep genomen. Van zijn buren stemmen de vierkantjes 9, 14 en 15 voor wit, en 22 voor zwart. De stembusuitslag luidt dus wit, maar vierkantje 1 was al wit en verandert dus niet. Nu vierkantje 2: twee van zijn buren stemmen voor wit en twee voor zwart, dus mag het zijn kleur behouden. Dan vierkantje 3: drie stemmen voor wit en een voor zwart; de uitslag is wit en het zwarte vierkantje 3 moet nu wit gekleurd worden (fig. 3). Bij vierkantje 4 staken de stemmen, maar vierkantje 5 moet weer veranderd worden (fig. 4).

Zo worden ook de overige twintig vierkantjes behandeld, en het blijkt daarbij dat de nummers 10, 12, 18, 20 en 22 nog van kleur moeten veranderen; dan heeft het gehele binnengebied een beurt gehad (fig. 5).

d. Moet het proces <u>stoppen</u>, nu elk vierkantje eenmaal aan de beurt geweest is? Er is geen enkele zekerheid dat een nieuwe ronde geen veranderingen meer zou opleveren. In ons vrij kleine voorbeeld (fig. 5) zou slechts één vierkantje nog van kleur veranderd worden (vierkantje nr. 6, zie fig. 6), maar in een groter tableau zouden waarschijnlijk veel meer veranderingen

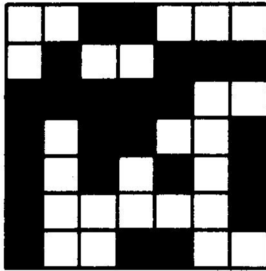


Fig. 1

Elk vierkantje is toevallig gekleurd

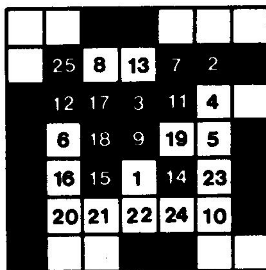


Fig. 2

Een willekeurige volgorde

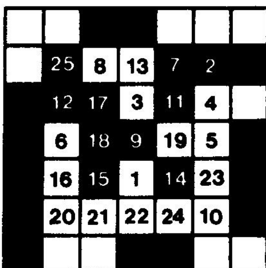


Fig. 3

Vierkantje 3 wordt wit

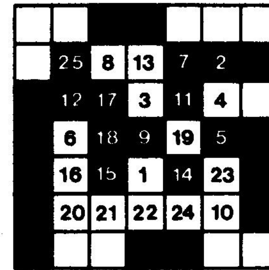


Fig. 4

Vierkantje 5 wordt zwart

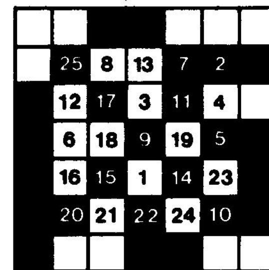


Fig. 5

Vierkantjes 10,12,16,20
en 22 veranderen

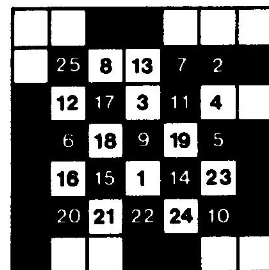


Fig. 6

In de tweede ronde verandert vierkantje 6 nog

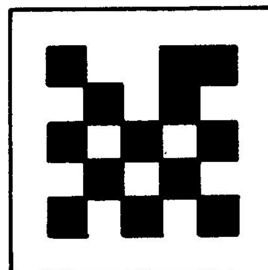


Fig. 7

Het resultaat

optreden. En ook na een tweede ronde valt er misschien nog iets te "verbeteren". Er moet een regel gevonden worden die een geschikt moment aangeeft om het proces te stoppen. Het is duidelijk dat het geen zin heeft door te gaan als een gehele ronde geen veranderingen meer heeft opgeleverd, maar het is niet altijd zeker dat dit ooit zal gebeuren. Het is namelijk heel goed mogelijk dat na een zeker aantal ronden een situatie bereikt wordt die al eens is voorgekomen, zodat het proces nooit een "natuurlijk" einde zou vinden. In de praktijk blijkt wel, dat het aantal veranderingen ronde na ronde afneemt, totdat deze daling stagneert en het aantal nu eens toe-dan weer afneemt. Besloten werd dat niveau van verandering te detecteren en zodra het bereikt werd te stoppen. Als signaal voor het bereiken van dit niveau wordt gebruikt: het gelijk blijven of stijgen van het aantal veranderingen per ronde.
Het stop-criterium is dus:

- het aantal veranderingen in de laatste ronde was nul

  of

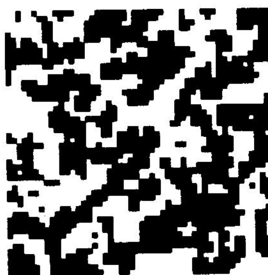
- het aantal veranderingen in de laatste ronde was groter dan of gelijk aan het aantal veranderingen in de voorlaatste ronde.

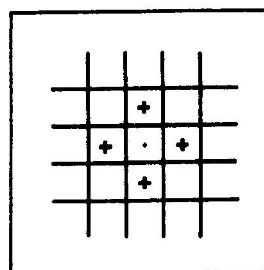
Het is duidelijk dat het proces nu op een zeker ogenblik zal stoppen, want het aantal veranderingen kan niet blijven dalen. In het voorbeeld stopt het proces, als bij de derde ronde geen veranderingen meer nodig blijken.
Fig. 7 toont het resultaat.

Het hierboven beschreven proces, dat de serie plaatjes in fig. 1 t/m 7 oplevert, is iets eenvoudiger dan het proces dat in feite geprogrammeerd is. Daarin worden niet alleen wit of zwart zonder meer (voor te stellen als -1 en +1) als kleuren gebruikt, maar zijn willekeurige positieve en negatieve getallen mogelijk om de mate van wit of zwart aan te geven.
Wel wordt begonnen met -1 en +1 als kleuren, maar bij het stemproces wordt de kleur van een vierkantje witter, naarmate de stembusuitslag witter was. In het getekende resultaat is van de mate van witheid en zwartheid niets meer te zien.

Resultaten

In figuur 8 worden de achtereenvolgende fasen getoond, die het kristallisatieproces doorloopt bij gebruik van het erbij gegeven recept 1:

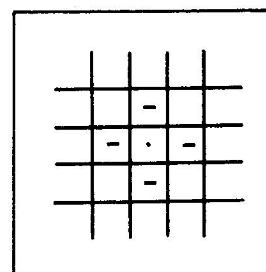
			
fase 0 (willekeurig beginpatroon)	fase 1 (na 1 ronde)	fase 2 (na 2 ronden)	fase 3 (na 3 ronden)
fase 4 (na 5 ronden)	fase 5 (na 7 ronden)	fase 6 (na 9 ronden)	fase 7 (resultaat, na 11 ronden)



recept 1

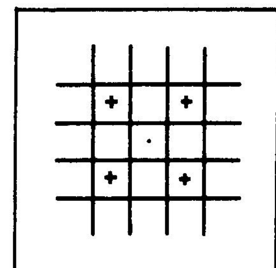
Fig. 8 Kristallisatie aan de hand van recept 1

Fig. 9 Kristalstruktuur geproduceerd
met recept 2



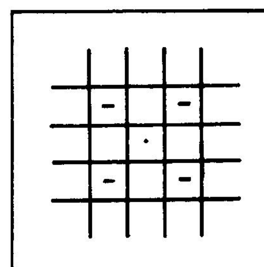
recept 2

Fig. 10 Kristalstruktuur geproduceerd met recept 3



recept 3

Fig. 11 Kristalstruktuur geproduceerd met recept 4



recept 4

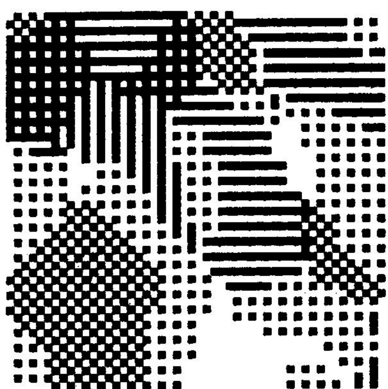
de vier directe buren stemmen elk volgens hun eigen kleur.
In de figuren 9, 10 en 11 zijn de eindresultaten te zien van het kristalliseren volgens drie andere eenvoudige recepten.
Figuur 12 geeft de resultaten van vier kristallisaties met wat ingewikkelder recepten.
De figuren 13 t/m 17 tonen het begin, enkele tussenstadia en het eindresultaat van een kristallisatie met een recept dat in figuur 12 ook al gebruikt is. Vergelijking van beide kristalstrukturen van gelijk recept leert dat dezelfde regelmaten in verschillende rangschikking voorkomen.

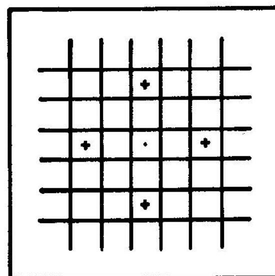
Andere mogelijkheden

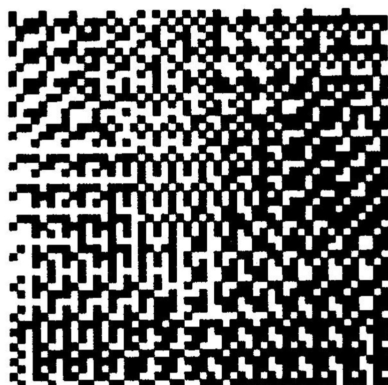
Het kristallisatie-idee kan op vele manieren gevarieerd worden.
Enkele mogelijkheden:

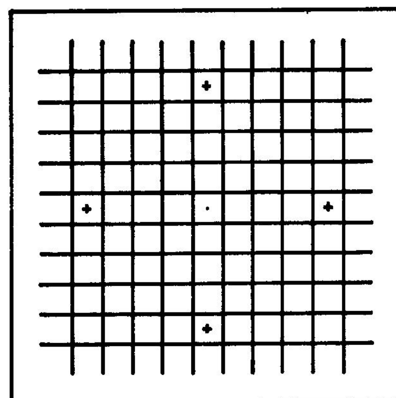
1. Meer dan twee kleuren. Elk vierkantje heeft dan niet een positief of negatief getal dat zijn kleur aangeeft, maar een vector die van elke kleur de sterkte aangeeft. Tenslotte is de grootste kleursterkte die in de vector voorkomt beslissend voor de kleur die het vierkantje krijgt.
2. Recepten die niet in de plus-en-min-vorm gegoten kunnen worden, b.v.:
   a. recepten waarin niet alleen de kleuren in de omgeving, maar ook de positie op het tableau een rol speelt;
   b. recepten als: een vierkantje wordt zwart, als van de 8 omringende vierkantjes er twee of drie zwart zijn, anders wordt het wit.
3. Niet alle omgevingsvierkantjes, die in het recept een rol spelen, stemmen, maar steeds één die door het lot wordt aangewezen.
4. Zoals de ontwikkelingsstadia van een kristalstruktuur een geleidelijke overgang te zien geeft, kan ook een metamorfose gecreëerd worden door, met gelijkblijvend recept, steeds kleine wijzigingen in het beginpatroon aan te brengen.
5. Idem, maar nu met kleine wijzigingen in het recept (waarbij i.p.v. + en - als "stemgedrag" de positieve en negatieve getallen toegelaten worden).

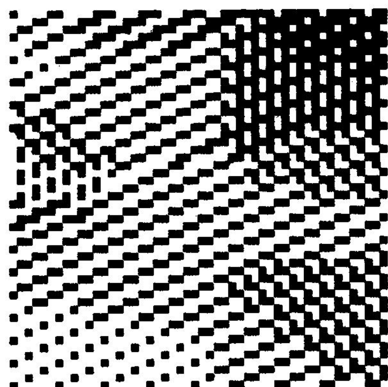
Een voor de hand liggende variatie van het beschreven kristallisatie-idee is een aanpak, waarbij het tableau niet vierkantje voor vierkantje (in een willekeurige volgorde) wordt veranderd, maar in zijn geheel: de nieuwe kleur van een vierkantje wordt dan niet in het tableau zelf aangebracht, maar in een apart tableau. De volgorde, waarin de vierkantjes geïnspecteerd

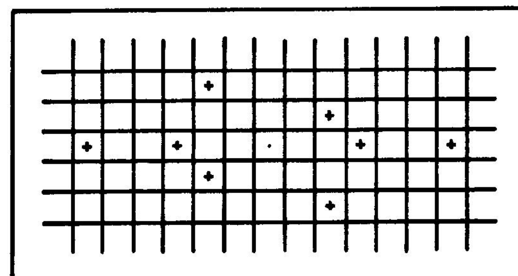


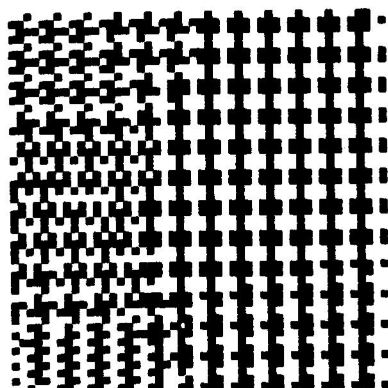












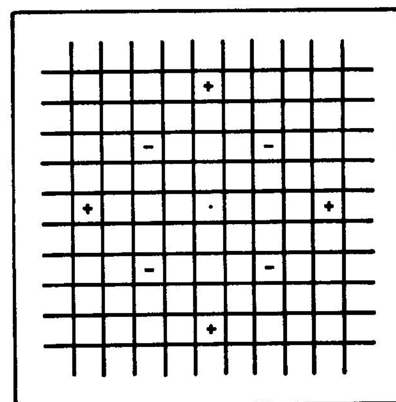
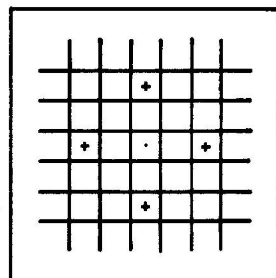


Fig. 12 Vier kristalstrukturen met recepten

Fig. 13 Willekeurige beginstruktuur voor kristallisatie aan de hand van recept 5



recept 5

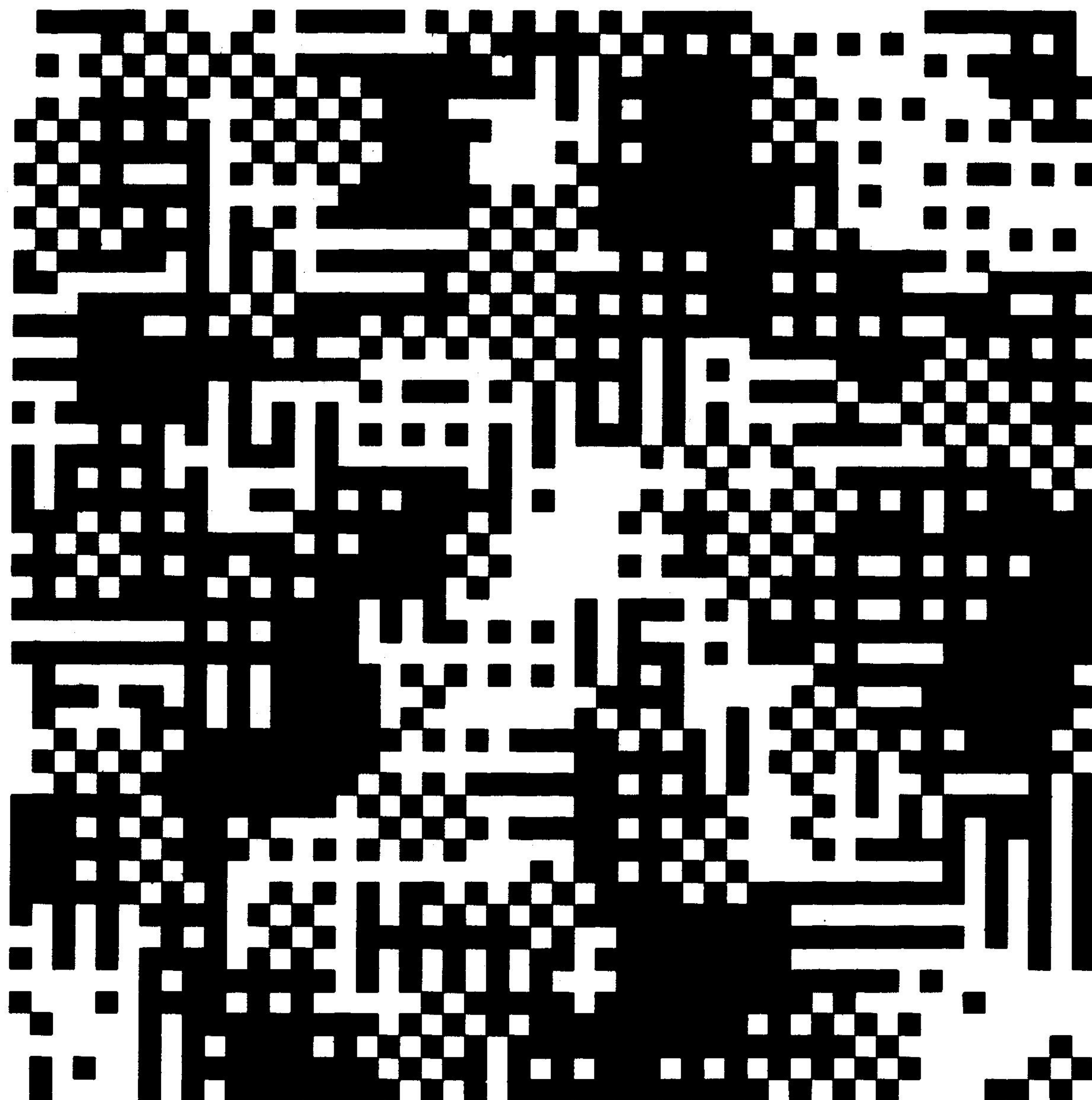


Fig. 14 Na 1 ronde

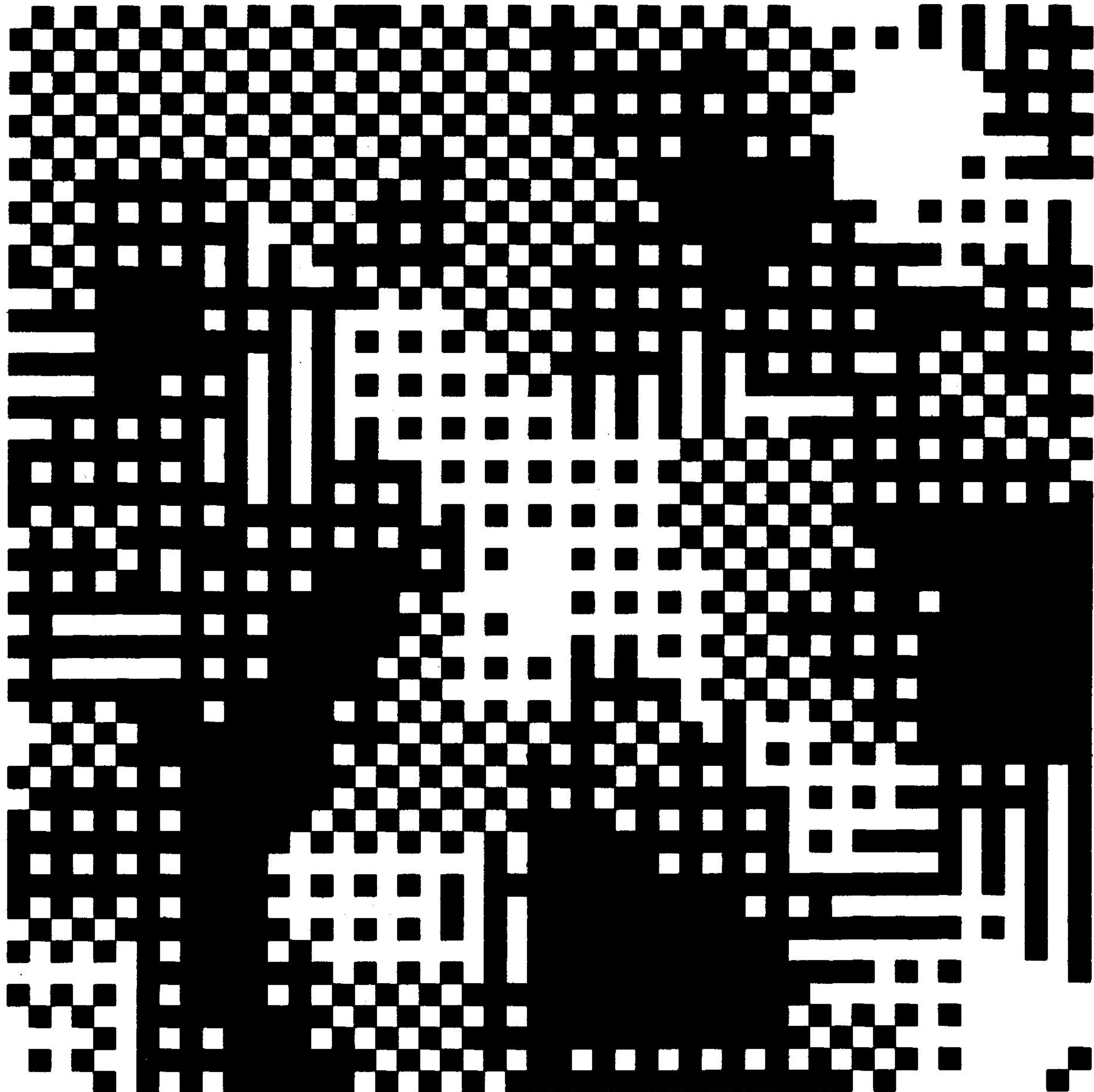


Fig. 15 Na 3 ronden

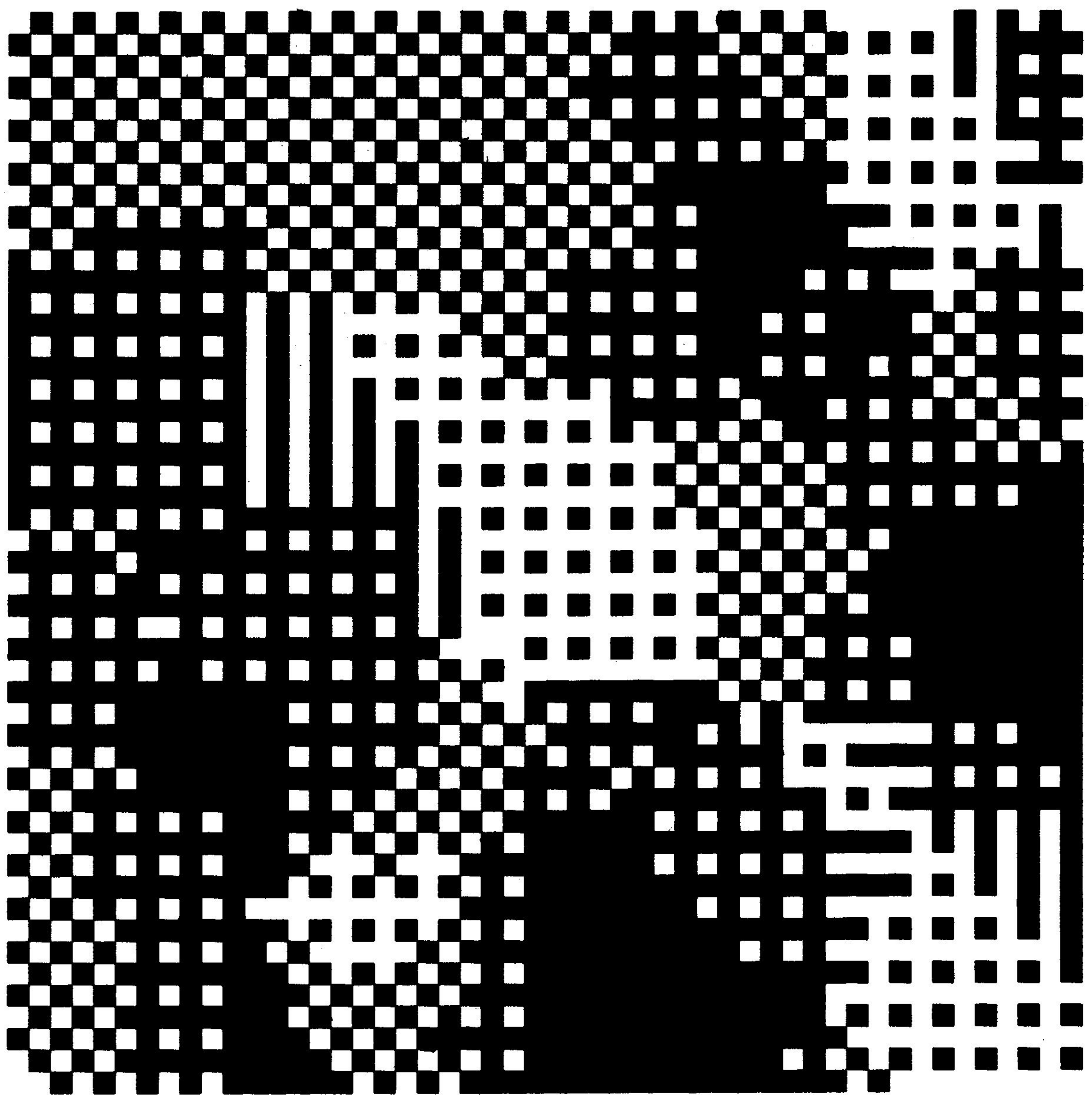


Fig. 16 Na 5 ronden

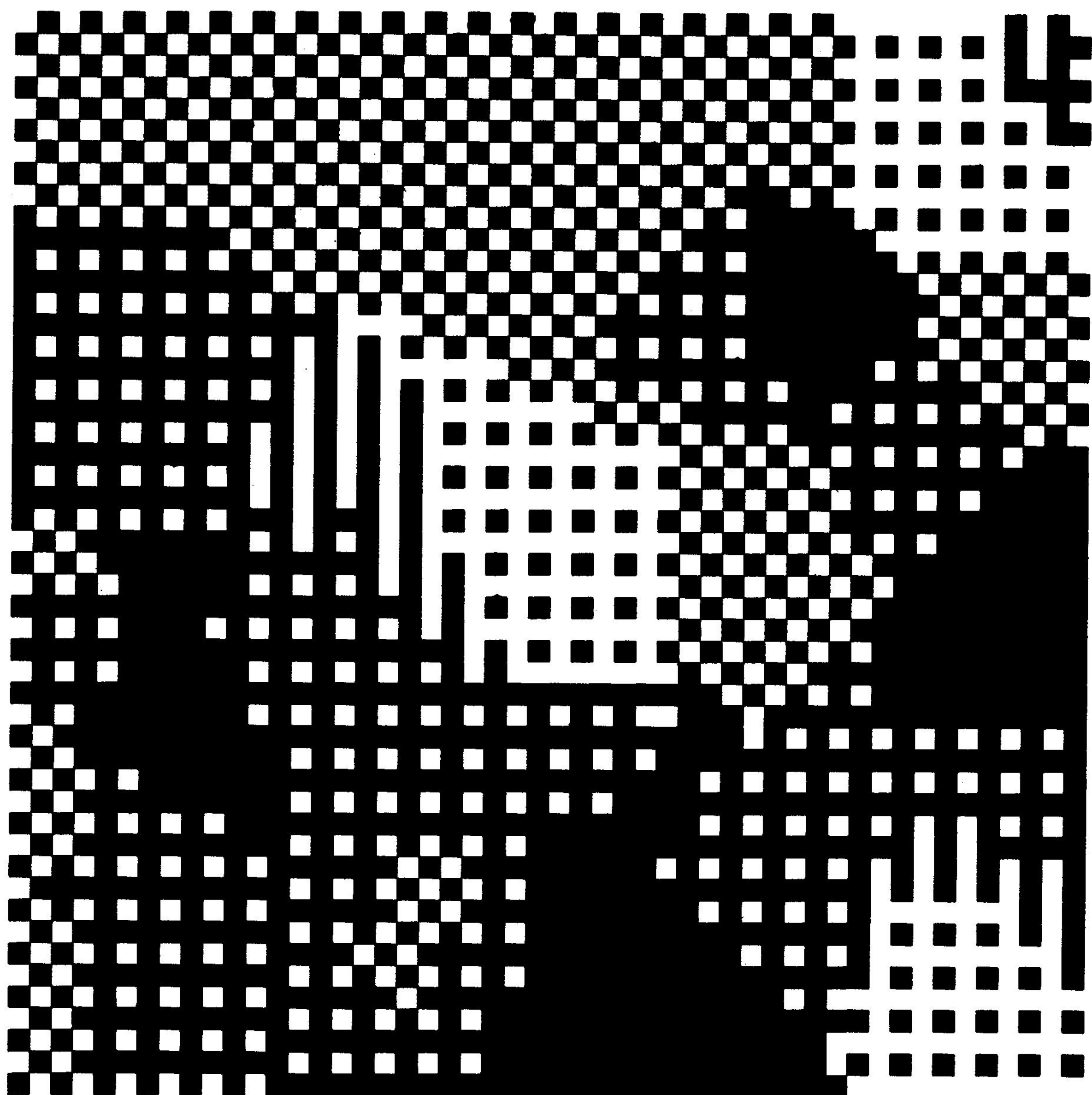


Fig. 17 Resultaat (na 8 ronden)

worden doet dan ook niet meer ter zake. Bij enig experimenteren bleek deze aanpak tot minder vloeiende scheidslijnen tussen de gebieden in het eindresultaat te leiden. In recepten, waarin min-tekens voorkomen, blijkt vaak geen convergentie op te treden: de vierkantjes, die in een zekere fase zwart zijn, zijn in de volgende wit en in de daarop volgende weer zwart.

Referenties

[1] *Eighth Annual Computer Art Contest,* Computers and Automation 19, 8 (1970), 13-24.

[2] Louis Andriessen, Leo Geurts, Lambert Meertens, *Componist en computer,* De Gids 132, (1969), 304-311.

[3] Leo Geurts, Lambert Meertens, *Kristalstrukturen,* Galerie Swart (1972).

[4] Leo Geurts, Lambert Meertens, *Kristalstrukturen,* Struktuur, een thema, een methode, ed. Hans Sizoo, Museum De Lakenhal, Leiden.

[5] L.G.L.T.Meertens, *Componeren met de computer,* Informatie 10, (1968), 418-421.

[6] L.G.L.T.Meertens, *Quartet no.1 in C major for two violins, viola and violoncello,* MR 96, Math.Centrum, (1968).

[7] L.G.L.T.Meertens, *Designing letter-like shapes,* Page (Bulletin of the Computer Arts Society) 17, July 1971.

[8] Nico Scheepmaker, *Computer als derde,* Avenue, sept. 1972, 118-121.